Resource Virtualization with End-to-End Timing Guarantees for Multi-Hop Multi-Channel Real-Time Wireless Networks

Jiachen Wang[†*], Tianyu Zhang[†], Xiaobo Sharon Hu[‡], Song Han[†]
[†]Dept. of Computer Science and Engineering, University of Connecticut, Storrs, CT, 06269
[†]Email: {jc.wang, tianyu.zhang, song.han}@uconn.edu
[‡]Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556
[‡]Email: shu@nd.edu

***Abstract*—Resource virtualization is a promising technique that has been increasingly deployed in industrial automation systems to support multiple time-critical applications sharing the same physical resources. Extensive studies have been reported on how to perform real-time virtualization on computing resources. However, when applying virtualization techniques on network resources (especially for real-time wireless networks), node dependency among applications, wireless channel contention and stringent end-to-end timing requirements of the real-time flows in the network pose severe challenges. To address this problem, this paper formulates the network virtualization problem for multi-hop multi-channel real-time wireless networks (RTWNs). We first present a Satisfiability Modulo Theory (SMT)-based exact solution to capture the constraints posted by each application's resource interfaces and node dependency graphs. A novel supply graph (SG)-based partitioning framework, SGP, is then proposed to determine the resource partitions for individual applications. SGP uses supply graph to maintain compliance with the regularity constraints while efficiently allocating resources. Experimental results from both a real-world testbed and extensive simulations show that SGP can achieve comparable success ratio with the SMT-based exact solution but reduce the computational overhead significantly.**

I. Introduction

Real-time wireless network (RTWN) technologies (e.g., WirelessHART [1], ISA100.11a [2], 6TiSCH [3], RT-WiFi [4], and other industrial solutions [5]–[8]) are becoming mature in recent years and have been widely deployed in industrial automation systems to support time-critical sensing and control applications. Traditional RTWNs are designed and deployed in a "vertical" way, where both network hardware and resource management mechanisms are highly customized to support specific application(s) [9]–[11]. However, with the ever-growing capability of hardware and complexity of applications, now it is possible and desirable to design a "one-fits-all" RTWN solution via resource virtualization, by deploying virtual networks for individual applications to share the same physical network resources but operate in an isolated fashion [12]–[14].

For example, a sensor-controller-actuator control loop (with stringent delay but low-bandwidth requirement) and a camera surveillance system (with high-bandwidth requirement but can tolerate larger delay) can be deployed on the same RTWN as long as their virtual networks are properly constructed. Such resource virtualization on RTWNs can greatly increase the utilization of the hardware, ease the resource management among multiple applications and improve system resilience. One application's fault(s) cannot propagate to other applications and thus avoid potential cascading failures in the system.

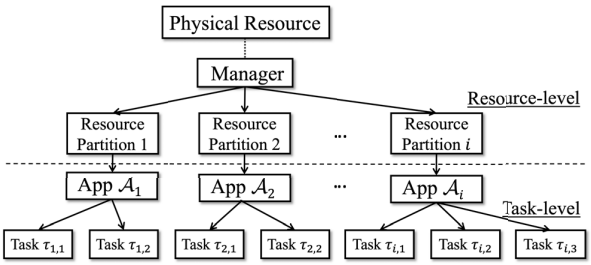


Fig. 1. Architecture of two-level real-time resource partitioning framework.

A key challenge in resource virtualization for RTWNs is how to achieve end-to-end timing guarantees for all the real-time flows in each application. This requires appropriate resource partitioning among individual applications to meet the timing requirements of their flows. Extensive studies have been reported in the literature on how to perform real-time virtualization on computing resources (e.g., CPU and GPU) using time and spatial partitioning, such as the Regularity-based Resource Partition (RRP) model [15], the Periodic model [16] and the Explicit Deadline Periodic (EDP) [17] model. Fig. 1 shows the typical architecture of the two-level real-time partitioning framework employed in these models. At the resource level, a resource manager divides the physical resources into a set of resource partitions in the time domain according to each application's timing requirements, and distributes the partitions to each application. At the task level, applications utilize their own scheduling policies to schedule the tasks on the received resource partitions. These models define resource interfaces to abstract the requirements (e.g., amount of resources, deadline, tolerable jitters) from each application's task set. As long as the allocated resource partitions satisfy the resource interfaces, the application can construct a feasible local schedule for its task set on its allocated resource partition(s).

Network resource virtualization in RTWNs bears some similarity with the aforementioned computing resource virtualization, but poses a unique challenge owing to the unavoidable sharing of network nodes by different applications. When constructing the communication schedule for an application in RTWN, each flow must access certain nodes in certain time

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

2023 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3503-2857-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/RTSS59052.2023.00040

2576-3172/23/$31.00 ©2023 IEEE
DOI 10.1109/RTSS59052.2023.00040

slots according to its *routing paths* and *timing requirements*. Partitioning the network resources while ignoring the node access control may cause node collision, where different applications require the same node to transmit packets at the same time. This node collision breaks the resource isolation among applications, which makes the allocated resource in some partitions unusable and thus may violate the timing guarantees for some flows. Some existing work on computing resource virtualization has considered sharing computing resources among inter-dependent applications. For instance, [18], [19] studies the time partitioning problem with access collisions on a multi-core and hardware accelerator platform, where tasks may compete for the shared GPUs. However, the focus of that work is on how to enforce time isolation on GPUs after the resource partitions are given. Its collision model is also much simpler than that of RTWNs, where tasks only request for some but not specific GPU resources. [20] improves the schedulability test of the RRP model considering intra-application dependent tasks. However, it does not consider inter-application task dependency, which is a key challenge in RTWN resource virtualization. [14] proposes an adaptive partition-based scheduling framework for 6TiSCH networks, and [21] proposes a hierarchical resource partitioning framework for dynamic resource management in industrial wireless networks. However, real-time requirements in these studies are not well considered. To the best of our knowledge, no existing work has studied the network resource virtualization problem in RTWNs with complex resource sharing patterns.

In this paper, we propose a network channel-node co-partitioning framework designed to address the unique challenges associated with RTWN resource virtualization. In addition to network channel resources, we also treat network nodes as a type of shared resource among applications and carefully partition application's access to them. By considering both inter- and intra-application node dependencies, the constructed network resource partitions are guaranteed to fulfill the end-to-end timing requirements of all applications. Specifically, we make the following contributions in this work.

- We formulate the network resource partitioning problem for multi-hop multi-channel RTWNs considering its unique constraints on both node and channel resources.
- We present a Satisfiability Modulo Theory (SMT)-based exact solution and a supply graph-based network virtualization framework to determine the network resource partitions.
- We implement the proposed network resource virtualization framework and evaluate the proposed algorithms through extensive experiments, including on a real-world 49-node 5-hop 6TiSCH testbed and a high-fidelity RTWN emulator.

The remainder of this paper is organized as follows. Section II describes the system model. Section III gives an overview of the proposed network resource virtualization framework. We formally define the resource partitions, interfaces and node dependencies and formulate the network resource partitioning problem in Section IV. The SMT-based exact solution and supply graph-based partitioning framework are presented in Section V and Section VI, respectively. Section VII summarizes the performance evaluation results through both testbed implementation and simulation studies. Section VIII concludes the paper and discusses the future work.

II. System Model and Problem Statement

In this section, we first present the RTWN system and application models employed in this work, and then define the network resource virtualization problem.

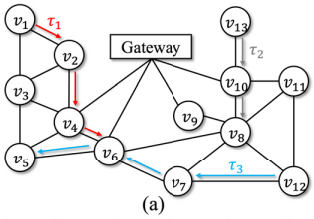
A. Network Model

We adopt a typical multi-hop multi-channel model used in many RTWNs (e.g., WirelessHART [1], ISA100.11a [2] and 6TiSCH [3]). A set of sensors, actuators and relay nodes are wirelessly connected to form a mesh topology, with a gateway sitting at the root of the network. The network is denoted as $G = (V, E)$, where $V = \{v_1, v_2, ...\}$ corresponds to the nodes and a link $e_{i,j} \in E$ represents a directed wireless communication from node v_i to node v_j. Each node is equipped with a single omnidirectional antenna operating in the half-duplex mode, where a node can only transmit or receive at most one transmission in each time slot. In the network, a centralized network manager is deployed on the gateway to manage and control the network, ensuring that all devices are operating properly. The network manager manages the network resources, including the access to each physical node and the transmission on each radio channel. It also performs resource allocation for individual applications according to their specific requirements.

In the network, a periodic end-to-end flow is represented as a *task*. Each task is defined by $\tau = (d, p, \psi)$, where d is the end-to-end deadline, p is the period of the task and $\psi = [v_1, v_2, ..., v_h]$ is the routing path consisting of h nodes. Each task releases an infinite sequence of instances, referred to as packets. In this paper, we assume a reliable RTWN [1] where the transmission of each hop requires only one time slot to complete. Thus, each packet requires $h - 1$ time slots to be transmitted from the source to the destination.

We utilize a multi-channel Time-Division Multiple Access (TDMA) based data link layer, in which time is divided into time slots, and consecutive slots are grouped into *slotframes* that repeat over time. We use T to denote the slotframe length. In each time slot, a total of C channels are available for packet transmissions. A schedule specifies the pair of nodes transmitting a packet on each channel in each time slot within the slotframe. The schedule is constructed by the network manager and installed on individual nodes. Fig. 2(a) shows an example 13-node mesh topology with one gateway and Fig. 2(c) shows a feasible schedule on a 10-slot 2-channel slotframe constructed for the three tasks shown in Fig. 2(b).

[1]Our solution can also be extended to support lossy links by taking redundant slots into consideration in the resource interfaces.



(a)

Task	p and d	ψ
τ_1	10	$v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_6$
τ_2	5	$v_{13} \rightarrow v_{10} \rightarrow v_8$
τ_3	10	$v_{12} \rightarrow v_7 \rightarrow v_6 \rightarrow v_5$

(b)

	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7	Slot 8	Slot 9	Slot 10
Ch 1	$e_{13,10} = \tau_2^1$	$e_{10,8} = \tau_2^1$	$e_{7,6} = \tau_3^1$	$e_{4,6} = \tau_1^1$	$e_{6,5} = \tau_3^1$	$e_{13,10} = \tau_2^2$	$e_{10,8} = \tau_2^2$			
Ch 2	$e_{12,7} = \tau_3^1$	$e_{1,2} = \tau_1^1$	$e_{2,4} = \tau_1^1$							

(c)

Fig. 2. (a) A RTWN mesh topology with 13 nodes and one gateway. (b) A task set with 3 tasks. (c) A schedule on a 10-slot 2-channel slotframe, where $e_{i,j} = \tau_m^k$ indicates that link $e_{i,j}$ is used to transmit a packet of the k-th instance of task τ_m.

B. Application Model

We assume that N time-critical applications $\mathcal{A}_i$ $(1 \leq i \leq N)$ are deployed on the RTWN. Each application $\mathcal{A}_i$ runs a periodic task set $\Gamma_i = \{\tau_{i,1}, \tau_{i,2}, , ..., \tau_{i,M_i}\}$, where each task $\tau_{i,k}|k \in [1, M_i]$ is associated with a period $p_{i,k}$, a deadline $d_{i,k}$, and a fixed routing path $\psi_{i,k}$ as described in the network model. Each application $\mathcal{A}_i$ applies a specific scheduling policy, denoted as SP_i, to schedule all the tasks in Γ_i. Each application requests a dedicated virtual 'sub-network' consisting of $C_i \leq C$ channels and a set of physical nodes in V. The resource associated with such sub-network must guarantee that the real-time requirements of all the tasks $\tau_{i,k} \in \Gamma_i$ are satisfied under the scheduling policy SP_i.

C. Problem Statement

Based on the above models, the objective of this work is to design a network resource virtualization framework for RTWNs to allow multiple time-critical applications to share the same physical network, and allow each application to use their own scheduling policy to schedule tasks on the allocated sub-network.

In the virtualization-enabled RTWNs, each application undertakes the network resource management for its own tasks by requesting a virtual sub-network and deploying a specific scheduling policy. The manager partitions the physical network into N sub-networks based on the request of each application $\mathcal{A}_i$. It must guarantee that $\mathcal{A}_i$ can satisfy the timing requirements of all its tasks in the allocated sub-network. This network virtualization problem can be defined as follows.

Problem 1 (Network Virtualization (NV))**.** *Given an RTWN with C channels and a set of applications $\mathcal{A}_i(1 \leq i \leq N)$, each of which employs a scheduling policy SP_i to schedule a set of periodic real-time tasks Γ_i, design a network virtualization framework to allow each application $\mathcal{A}_i$ to schedule its tasks using SP_i in a virtual sub-network while guaranteeing the timing requirements of all the tasks.*

To solve the NV problem, an intuitive idea is to let each application submit all the task specifications to the network manager. Then, the manager generates a global schedule for all the applications and comprises the resource allocated to each application in the global schedule as the virtual sub-network. Such a method, however, cannot be applied to solve the NV problem due to the following three reasons.

First, a time-critical application deployed in a RTWN may not be willing to release all the task specification information for security and privacy considerations. Instead, it desires to use a resource interface to effectively abstract the required sub-network resources to satisfy the tasks' real-time requirements. Second, each application employs an individual scheduling policy which prevents the network manager to directly apply any existing global scheduling algorithms (e.g., [22], [23]) where a common scheduling policy is deployed to schedule all the tasks running in the network. Third, the NV problem requires the network manager to allocate resource at the application level (i.e., generating a sub-network), while guaranteeing the timing requirements at the task level (i.e., satisfying the task deadlines using the sub-network). Therefore, a global scheduling framework which requires all the task specification information and performs resource allocation at the task level cannot solve the NV problem.

III. Network Virtualization Framework Overview

A. The NV Problem Challenges

The NV problem can be decomposed into two sub-problems: i) how to determine the sub-network request (i.e., designing the resource interface) at each application, and ii) how to create the sub-networks for individual applications (i.e., performing resource partitioning) at the network manager to satisfy the functional and timing requirements of all the applications. The sub-network requested (allocated) by each application (the network manager) can be captured by the network resources in each hyperperiod. The network resources can be classified into i) **channel resources** consisting of C communication channels, and ii) **node resources** consisting of all the nodes in the network. These two types of resources in a RTWN are subject to the following two constraints that pose the key challenges for solving the NV problem.

Constraint 1 (Channel capacity)**.** *In any time slot, the number of channels allocated to all the applications cannot be larger than C.*

Constraint 2 (Node access)**.** *Any half-duplex node cannot transmit/receive more than one packet in any time slot. That is, node collision occurs if two applications access the same node for transmission in the same time slot.*

The channel capacity constraint implies that we can parallelize the execution of multiple applications to improve the channel efficiency as long as $\sum_i C_i \leq C$. However, the node access constraint indicates that parallel applications may cause failed transmissions due to node collision and degrade the channel efficiency. These two opposite observations pose the first challenge of the NV problem.

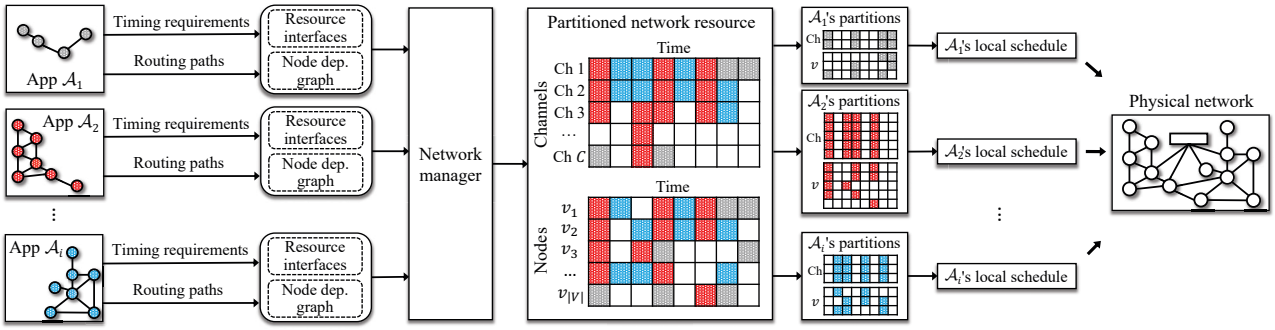


Fig. 3. An overview of the execution model for the proposed network resource virtualization framework in RTWNs.

Challenge 1. How to avoid node collision among the transmissions from different applications while accommodating more parallel applications to improve the channel efficiency.

To tackle Challenge 1, at the highest level, we aim to perform resource request and partitioning in terms of channel resources and node resources independently. In this way, we can avoid node collision by preventing each node from being used by two applications with concurrent channel access. Since node collision only occurs when a node is requested by multiple applications, we define such node as *critical node*, and perform resource request/partitioning for all critical nodes.

Definition 1 (Critical Node)**.** *A critical node v_j^* is a node on the routing paths of tasks from more than one application.*

Specifically, we aim to determine the channel resources (i.e., the time slots within each of which application $\mathcal{A}_i$ can access C_i channels) and the node resources for each critical node v_j^* (i.e., the time slots that $\mathcal{A}_i$ can access v_j). Since one transmission can be transmitted in a time slot only if both the channel resource and the corresponding critical node resource (if needed) are available[2], we must guarantee that the overlap of these two types of resources (i.e., the common set of time slots) can meet the timing requirements of individual applications, i.e., the deadlines of each task in the application.

Furthermore, given that the transmission of each task needs to follow the pre-determined routing path, the node resource allocated to each application must be compliant with the tasks' routing paths. That is, the access to the critical nodes in each application must follow a specific sequence. Otherwise, some allocated critical node resources can be wasted and the timing requirement of the corresponding task can not be met. Thus, this particular node dependency requirement poses another challenge for the NV problem.

Challenge 2. How to guarantee that the critical node resources allocated to each application satisfy the node dependency requirements of all the tasks running in the application.

To tackle Challenge 2, we employ a dependency graph to capture the relationship between the critical nodes in each application. The dependency graph can be generated by each application and submitted to the network manager as a part of the sub-network request. The network manager then partition node resources for each application according both the resource interfaces and the node dependency graph.

[2]Since the other non-critical nodes are not shared by any applications, each application can access any non-critical node in a time slot if the channel resource is available.

TABLE I
SUMMARY OF IMPORTANT SYMBOLS AND NOTATIONS.

Symbol	Description
$G = (V, E)$	Network topology, V: nodes set, E: links set
T	Slotframe length
C	Total number of available channels in the network
$\mathcal{A}_i$	The i-th application deployed on the network
Γ_i	Task set of $\mathcal{A}_i$
C_i	Number of required channels of $\mathcal{A}_i$
$P_i^{\mathcal{R}}$	Resource partition of $\mathcal{A}_i$ for resource $\mathcal{R}$
$\alpha(P)$	Availability factor of partition P
$r(P)$	Supply regularity of partition P
$I_i^{\mathcal{R}} = (\alpha, r)$	Resource interface of $\mathcal{A}_i$ for resource $\mathcal{R}$
$S(P, t)$	Supply function of partition P
$ir(P, t)$	Instant regularity of partition P at time t
$\mathcal{G}_i^P$	Node precedence dependency graph of $\mathcal{A}_i$
$\mathcal{G}_i^C$	Node transmission conflict graph of $\mathcal{A}_i$

B. Execution Model

The high-level execution model of the network resource partitioning framework is shown in Fig. 3. Each application $\mathcal{A}_i$ initializes a sub-network request with channel/node resource interfaces and a node dependency graph, to abstract the resource requirements according to its tasks' specification. After receiving the sub-network requests from all the applications, the network manager determines the sub-networks, in terms of channel and critical node partitions, allocated to each application, and send them back to the requesting applications. Upon receiving the channel and critical node partitions, each application $\mathcal{A}_i$ follows its own scheduling policy to generate a local schedule for the task set Γ_i and deploys the schedule on the corresponding nodes in the physical network.

Based on this network resource virtualization framework, we need to design i) the channel/node resource interfaces and ii) the node dependency graph at individual applications, and iii) the channel/node partitions at the network manager to meet the timing requirements of all the tasks running in each application. The details of each component are described in the following sections.

IV. Resource Interface Design

We first discuss how to design the channel/node resource interfaces and construct the node dependency graph to be submitted by each application to the network manager.

A. Preliminaries

A resource interface captures the network resources requested by an application. A feasible resource interface must guarantee that the resources allocated by the network manager according to the resource interface can satisfy the real-time requirements of the tasks running in the application.

The Regularity-based Resource Partitioning (RRP) model proposes to use a combination of *availability factor* and *regularity* to define the resource interface to capture the temporal resource (processor) requirement of each application [24], [25]. The RRP-based resource interface provides schedulability guarantees under various scheduling policies. Thus in this paper, we develop channel/node resource interfaces based on the RRP model to capture application's timing requirements on individual channel and critical nodes. We first introduce some definitions used in the RRP model.

Definition 2 (Resource Partition)**.** *A resource partition (or partition for short) $P_i^{\mathcal{R}} = \{s_1, s_2, ..., s_n\}$ consists of a set of time slots within the slotframe, and application $\mathcal{A}_i$ can access resource $\mathcal{R}$ in those time slots in $P_i^{\mathcal{R}}$.*

Since we manage the channel resources and node resources independently, the resource $\mathcal{R}$ can be C_i network channels or a critical node used by application $\mathcal{A}_i$.

Definition 3 (Availability Factor)**.** *The availability factor of a partition P, denoted as $\alpha(P) = |P|/T \in [0, 1]$, represents the fraction of the time slots allocated to P in each slotframe.*

Definition 4 (Supply Function)**.** *Supply function $S(P, t)$ of a partition P represents the number of allocated time slots in interval $[0, t]$, i.e ., $S(P, t) = |\{s \in P | s \leq t\}|$.*

Definition 5 (Instant Regularity)**.** *The instant regularity of a partition P at time t is defined as $ir(P, t) = S(t) - \alpha(P) \cdot t$, which captures the difference between the number of supplied slots by t and the expected number of slots to be uniformly allocated by t according to the availability factor $\alpha(P)$.*

Definition 6 (Supply Regularity)**.** *The supply regularity SR of a partition P is defined as the maximum value of $r(P) = |ir(P, t_1) - ir(P, t_2)|$, for all $t_1, t_2 \in [1, T]$.*

The supply regularity serves as an upper bound on the deviation between the actual resource supply during any time interval and the expected resource supply that the partition is intended to receive. It plays a crucial role in determining the slot allocations for the partitions in our supply graph based partitioning framework to be elaborated in Section VI.

Fig. 4 shows an example of the supply function and instant regularity of partition $P = \{4, 7, 9\}$ in a 10-slot slotframe. The availability factor of this partition is $\alpha(P) = 3/10 = 0.3$. The maximum instant regularity is $ir(P, 9) = 3 - 9 \times 0.3 = 0.3$, and

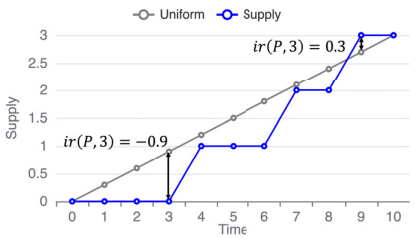


Fig. 4. The supply function of an example partition $P = \{4, 7, 9\}$ with slotframe length $T = 10$. We have $\alpha(P) = 0.3$ and $r(P) = 1.2$.

the minimum instant regularity is $ir(P, 3) = 0 - 3 \times 0.3 = -0.9$. Thus, the supply regularity of partition P is $r(P) = |ir(P, 9) - ir(P, 3)| = 1.2$.

Definition 7 (Resource Interface)**.** *The resource interface of application $\mathcal{A}_i$ is defined as $I_i^{\mathcal{R}} = (\alpha, r)$, where α and r are the availability factor and the regularity of the requested partition on resource $\mathcal{R}$, respectively.*

According to [24], [26], the timing requirements of all the tasks can be guaranteed if the resource partition $P_i^{\mathcal{R}}$ allocated to application $\mathcal{A}_i$ satisfies the resource requirements specified by its resource interface $I_i^{\mathcal{R}}$, i.e., $\alpha(P) \geq \alpha$ and $r(P) < r$.

Given a task set Γ_i and a scheduling policy SP_i, the RRP model [24], [26] provides the schedulability test for Γ_i running on a single processor partition specified by α and r under SP_i. We denote I_i^C and $I_i^{v^*}$ as the channel interface and node interface for critical node v^*, respectively. Below we describe how to determine the channel/node resource interfaces, i.e., α and r of I_i^C and $I_i^{v^*}$, for task set Γ_i running in $\mathcal{A}_i$.

B. Resource Interface

1) Node Resource Interface: Since each critical node is equivalent to a single processor, we can directly rely on the schedulability test provided in the RRP model to determine each critical node interface $I_i^{v^*}$. Below, we introduce the schedulability test for the EDF policy as an example. Readers can refer to [24], [26] for more details and the schedulability tests for other commonly used scheduling policies.

Theorem 1. *Task set $\Gamma = \{\tau_i = (c_i, p_i) | i = 1..n\}$ is schedulable on a partition P by EDF if $\sum_{i=1}^{n} c_i/(p_i - \lceil (r-1)/\alpha \rceil) \leq \alpha$, where c is the execution time of the task [26].*

Since we are only concerned with the resource demand for critical node v^* rather than for all the nodes, task set Γ should be the subset of Γ_i that only contains the transmissions passing through v^*. That is, two notable modifications are needed when we apply Theorem 1 to determine the node interface $I_i^{v^*}$ for critical node v^*. First, task set Γ only consists of the tasks having critical node v^* on their routing paths. Second, the execution time of each task in Γ is either 1 or 2, if v^* is the source/destination node or a relay node on the routing path of the original task in Γ_i.

Based on the above discussion, we can first determine the availability factor of task set Γ according to the number of required slots in each slotframe. That is, $\alpha = \sum_{\tau_i \in \Gamma} c_i/p_i$.

Then based on Theorem 1, the largest r satisfying the inequation is picked as the regularity since a larger regularity provides higher flexibility to the partitioning.

2) Channel Resource Interface: Determining the channel resource interface I_i^C for each task set Γ_i is more challenging since each application $\mathcal{A}_i$ requires C_i channels and the extension of the schedulability test (e.g., Theorem 1) to multi-processor platforms is non-trivial, especially the multi-hop transmission turn the tasks into DAG tasks [27], [28]. As an alternative, we propose a schedule-based approach to determine the regularity and availability factor of the channel resource interface for Γ_i.

Specifically, we let application $\mathcal{A}_i$ generate a schedule for task set Γ_i by running scheduling policy SP_i on a network with C_i channels. Suppose the number of slots within the slotframe in the generated schedule is a. An intuition to set the availability factor for Γ_i is $\alpha = a/T$, which is the same setting for the node interface described above. However, setting $\alpha = a/T$ cannot satisfy the timing requirement of Γ_i since a is generated assuming that $\mathcal{A}_i$ is the only application running in the network without any critical node sharing with other applications. Therefore, we let each application require an additional set of redundant slots to accommodate the potential unavailability of certain slots due to critical node contention with other applications. Since each application has no knowledge of the critical node requirement from other applications, the setting of such redundancy is empirical that an additional time slot is added to the channel resource interface[3].

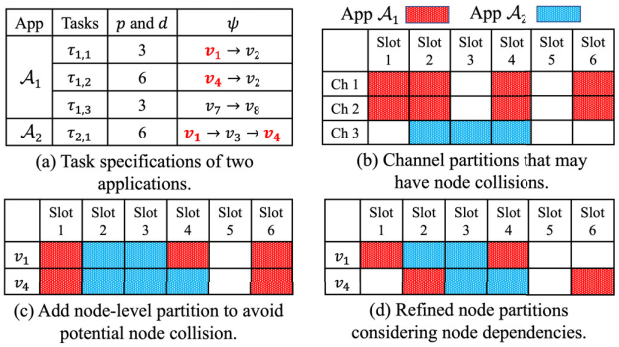
After the availability factor α is determined, we set regularity $r = 1$ for the channel resource interface of each application since $r = 1$ is a safe regularity lower bound to guarantee the schedulability of the task set according to the RRP model. The example below demonstrates the channel and node resource interfaces determined for the application in Fig. 2.

Example 1. *Consider application $\mathcal{A}_i$ in Fig. 2(b). Suppose node v_6 is a critical node shared by other applications. To determine $I_i^{v_6}$, we identify the tasks passing through v_6 (i.e., τ_1 and τ_3) and determine the execution time of each task (i.e., $c_1 = 1$ and $c_3 = 2$). Thus, the availability factor of node resource interface for v_6 equals to $\frac{1}{10} + \frac{2}{10} = 0.3$. Then, according to Theorem 1, the largest feasible regularity is 1.6. To determine the channel resource interface, $\mathcal{A}_i$ generates a schedule as shown in Fig. 2(c) using EDF. Interface $I_i^{C_i}$ is then set to $(\alpha = \frac{7+1}{10} = 0.8, r = 1)$ where an additional redundant slot is considered.*

C. Intra-Application Node Dependency

As described in Challenge 2 in Section III-A, the critical node resources allocated to each application must satisfy the node dependency requirements of all the tasks running in the application. In this section, we describe how to handle the node dependency by using the proposed node dependency graph.

[3]Although the network manager has the information of critical nodes' requirements from all the applications, it is unaware of the specific timing requirement of each application, thus cannot determine a proper redundancy setting for each application.



(a) Task specifications of two applications.

(b) Channel partitions that may have node collisions.

(c) Add node-level partition to avoid potential node collision.

(d) Refined node partitions considering node dependencies.

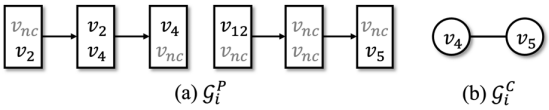
Fig. 5. An example showing the node dependency. (a) v_1 and v_4 are the critical nodes shared by $\mathcal{A}_1$ and $\mathcal{A}_2$. (b) Channel partitions for $\mathcal{A}_1$ and $\mathcal{A}_2$. (c) Critical node partitions. (d) Refined critical node partitions with less resources.

Example 2. *The example in Fig. 5 illustrates how the node dependency impacts the resource partitions. Consider two applications in Fig. 5(a) and the channel partitions allocated to the applications according to their channel resource interfaces are shown in Fig. 5(b). Specifically, Application $\mathcal{A}_1$ is allocated with channels $\{1, 2\}$ and slots $\{1, 2, 4, 7\}$. Application $\mathcal{A}_2$ is allocated with channel 3 and slots $\{2, 3, 4\}$. To avoid potential critical node collision among the applications, a node partition for each critical node is allocated to each application as shown in Fig. 5(c).*

However, the critical node partition without considering the node dependency requirements can cause resource waste. For example, v_1 and v_4 transmit to the same node v_2 in application $\mathcal{A}_1$, thus these two nodes cannot transmit simultaneously since v_2 can receive packets from at most one sender in one time slot. Then, one of the node resources allocated to $\mathcal{A}_1$ in slot 1 will be wasted. Therefore, v_1 and v_4 should not be allocated to $\mathcal{A}_1$ in the same slot and Fig. 5(d) shows the refined critical node partition where less resources are used in the critical node partitions allocated to the two applications.

We define two types of node dependency graphs to represent the relationship among critical nodes. The first type, named *node precedence graph*, captures the precedence of critical nodes in the tasks' routing paths. The second type, named *node conflict graph*, captures the constraint that two critical nodes cannot transmit to (receive from) the same receiver (sender) simultaneously, as shown in Example 2.

Definition 8 (Node Precedence Graph)**.** *Node precedence graph $\mathcal{G}_i^P$ is a set of disconnected Directed Acyclic Graphs (DAGs) for application $\mathcal{A}_i$, denoted as $\{\mathcal{D}_{i,j|j=1,2,\dots}\}$. Each DAG $\mathcal{D}_{i,j} = (X_j, E_j, p)$ captures the precedence imposed by a task $\tau \in \Gamma_i$ passing through at least two critical nodes. Each vertex $x = (v_j, v_k) \in X_j$ represents a transmission from sender v_j to receiver v_k in τ where at least one of v_j and v_k is a critical node. Each directed edge $e = (x_p, x_d) \in E_j$ represents a precedence between two vertices x_p and x_d. p is the period of $\mathcal{D}_{i,j}$ and equals to the period of τ.*



(a) $\mathcal{G}_i^P$ (b) $\mathcal{G}_i^C$

Fig. 6. Node dependency graph example for the task set in Fig. 2(b) supposing that v_2, v_4, v_5, v_{12} are critical nodes. (a) Node precedence graph. (b) Node conflict graph.

Since we are only interested in the critical nodes, we use v_{nc} to denote any non-critical node in each vertex in the node precedence graph. Node precedence graph introduces the following two constraints.

Constraint 3. *If two critical nodes are in the same vertex of a DAG $\mathcal{D}$, they should be allocated in the same time slot.*

That is, for each vertex $x = (v_j, v_k)$ in node precedence graph $\mathcal{G}_i^P$, partition $P_i^{v_j}$ must have a common slot with partition $P_i^{v_k}$ in every p slots, where p is the period of $\mathcal{D}$.

Constraint 4. *Nodes in predecessor vertices of a DAG $\mathcal{D}$ must be allocated earlier than the nodes in descendant vertices.*

That is, for each edge $e = ((v_j, v_k), (v_l, v_m))$ in the node precedence graph $\mathcal{G}_i^P$, partitions $P_i^{v_j}$ and $P_i^{v_k}$ must have a slot earlier than partitions $P_i^{v_m}$ and $P_i^{v_n}$ in every p slots, where p is the period of $\mathcal{D}$.

Definition 9 (Node Conflict Graph)**.** *In node conflict graph $\mathcal{G}_i^C = (V, E)$, each vertex $v \in V$ is a critical node used by application $\mathcal{A}_i$ and each edge $e = (v_j, v_k), e \in E$ indicates that nodes v_j and v_k conflict.*

An edge $e = (v_j, v_k)$ in the node conflict graph $\mathcal{G}_i^C$ represents that v_j and v_k cannot be allocated in the same slot since they have a same sender or receiver node. That is, partition $P_i^{v_j}$ cannot have a common slot with partition $P_i^{v_k}$.

The node precedence graph and node conflict graph can be readily derived according to the task set specification of each application. Fig. 6 shows the node precedence graph and node conflict graph derived from the task set in Fig. 2(b). Suppose v_2, v_4, v_5, and v_{12} are the critical nodes. Since two tasks τ_1 and τ_3 pass through the critical nodes, the node precedence graph $\mathcal{G}_i^P$ consists of two DAGs. Since node v_6 is a common peer node, i.e., the receiver of v_4 and the sender of v_5, we connect nodes v_4 and v_5 in the node conflict graph to avoid them from transmitting/receiving to/from v_6 simultaneously.

D. Network Resource Partitioning Problem

Given the resource interfaces and node dependency graphs of all the applications, the network manager needs to generate the partition for each application, satisfying the resource requirements specified by the resource interfaces. We give the network resource partitioning problem formulation below.

Problem 2 (Network Resource Partitioning (NRP))**.** *Given a RTWN with C channels and a set of applications $\mathcal{A}_i (1 \leq i \leq N)$, each with a channel resource interface $I_i^{C_i}$ and a set of critical node interfaces $I_i^v (v \in V^C)$, node precedence graph $\mathcal{G}_i^P$ and node conflict graph $\mathcal{G}_i^C$, determine the channel partition $P_i^{C_i}$ and the critical node partitions $P_i^v (v \in V^C)$ for each application while satisfying the following constraints:*

Constraint 5. *Each channel and each node can only be allocated to at most one partition in any time slot.*

Constraint 6. *Each resource partition $P_i^{\mathcal{R}}$ satisfies the resource requirements specified by resource interface $I_i^{\mathcal{R}}$, i.e., $\alpha(P) \geq \alpha$ and $r(P) < r$.*

Constraint 7. *Node partitions of application A_i satisfy the constraints specified by the node precedence graph $\mathcal{G}_i^P$ and node conflict graph $\mathcal{G}_i^C$.*

Constraint 8. *The number of channels allocated to all the applications in each time slot cannot be larger than the total number of channels in the network.*

V. SMT-based Exact Solution

To solve the NRP problem, we formally present a set of constraints in this section on constructing feasible resource partitions for all the applications while satisfying their resource interfaces and node dependency graphs. A Satisfiability Modulo Theory (SMT) solver can be applied to find an exact solution if exists.

Variables: We use three inter-convertible variables to represent the partition allocation of each application.

i) $x_i^{\mathcal{R}}(t)$ is a Boolean variable indicating whether resource $\mathcal{R}$ is allocated to application $\mathcal{A}_i$ in time slot t.
ii) $s_{i,j}^{\mathcal{R}} \in [1, T]$ denotes the index of the j-th time slot ($j \in [1, \alpha \cdot T]$) in partition $P_i^{\mathcal{R}}$.
iii) $S(P_i^{\mathcal{R}}, t)$ denotes the number of allocated time slots in partition $P_i^{\mathcal{R}}$ within the time interval $[0, t]$.

These three variables are inter-convertible as below.

$$S(P_i^{\mathcal{R}}, t) = \begin{cases} S(P_i^{\mathcal{R}}, t-1) + 1, & \text{if } x_i^{\mathcal{R}}(t) \text{ is } \textit{True}, \\ S(P_i^{\mathcal{R}}, t-1), & \text{otherwise.} \end{cases} \quad (1)$$

$$s_{i,j}^{\mathcal{R}} = t \text{ if } S(P_i^{\mathcal{R}}, t) = j \text{ and } S(P_i^{\mathcal{R}}, t-1) = j - 1. \quad (2)$$

Mutually exclusive constraint: In each time slot, any resource $\mathcal{R}$ can be allocated to at most one application's partition to avoid access collision among applications.

$$\forall t \in [1, T], \forall \mathcal{R} : \sum_{i \in [1, N]} x_i^{\mathcal{R}}(t) \leq 1. \quad (3)$$

Partition supply constraint: The number of slots in each resource partition $P_i^{\mathcal{R}}$ in a slotframe must be larger than or equal to the number of slots specified by the availability factor of the corresponding resource interface $I_i^{\mathcal{R}} = (\alpha, r)$.

$$\forall i \in [1, N], \forall \mathcal{R} : S(P_i^{\mathcal{R}}, T) \geq \alpha \cdot T. \quad (4)$$

Partition regularity constraint: The regularity of resource partition $P_i^{\mathcal{R}}$ must satisfy the regularity requirement specified by the corresponding resource interface $I_i^{\mathcal{R}} = (\alpha, r)$.

$$\forall i \in [1, N], \forall \mathcal{R}, \forall t_1, t_2 \in [1, T], t_1 \neq t_2 : \\ | \left(S(P_i^{\mathcal{R}}, t_1) - \alpha \cdot t_1\right) - \left(S(P_i^{\mathcal{R}}, t_2) - \alpha \cdot t_2\right) | < r. \quad (5)$$

Resource availability constraint: A transmission involving any critical node can be transmitted only if a channel resource is available in the same time slot.

$$\forall i \in [1,N], \forall v \in V^i \cap V_C, \forall j_1, \exists j_2 : s^v_{i,j_1} = s^{C_i}_{i,j_2}. \quad (6)$$

Channel capacity constraint: The total number of channels allocated to all the applications cannot be larger than C.

$$\forall t \in [1,T] : \sum_{i \in [1,N]} C_i \cdot x_i^{C_i}(t) \leq C. \quad (7)$$

Node precedence constraint: Let $o_{i,j} = \{x | x \in X_j\}$ denote the topological ordering of precedence DAG $D_{i,j} = (X_j, E_j, p)$. We denote $d_{i,j,k}(h)$ as the time slot allocated to the h-th transmission vertex in $o_{i,j}$ during its k-th ($k \in [1, T/p]$) repetition within the slotframe. We have:

$$\forall i \in [1,N], \forall D_{i,j} = (X_j, E_j, p) \in \mathcal{G}_i^P,$$
$$\forall k \in [1, T/p], \forall h \in [1, |X_j|], \forall v \in o_{i,j}[h] :$$
$$\sum_{m \in [1, P_{i,j}]} s^v_{i,m} = d_{i,j,k}(h) = 1. \quad (8)$$
$$1 + (k-1) \cdot p \leq d_{i,j,k}(h) \leq k \cdot p. \quad (9)$$
$$d_{i,j,k}(h) < d_{i,j,k}(h+1). \quad (10)$$

Eq. (8) maps the variable $d_{i,j,k}(h)$ to the allocated time slots in each resource partition. Eq. (9) specifies the time duration for the DAG. Eq. (10) formulates the precedence according to the topological ordering of the DAG. In addition, we need to ensure that each allocated slot in a partition is used to satisfy at most one vertex in the DAG. That is,

$$\forall i \in [1,N], \forall j_1, j_2, \forall k_1, k_2, \forall h_1, h_2, \forall v \in V :$$
$$\neg(v \in o_{i,j_1}(h_1) \cap o_{i,j_2}(h_2)) \wedge (d_{i,j_1,k_1}(h_1) \neq d_{i,j_2,k_2}(h_2))) \quad (11)$$

Node conflict constraint: Conflicting nodes in node conflict graph $\mathcal{G}_i^C = (V, E)$ must not be accessed by application $\mathcal{A}_i$ in the same time slot. That is,

$$\forall i \in [1,N], \forall v_1, v_2 \in V, \forall t \in [1,H] :$$
$$\neg(x_i^{v_1}(t) \wedge x_i^{v_2}(t)) \wedge (e(v_1, v_2) \in E)) \quad (12)$$

Based on the above constraints, an SMT formulation can be derived to find an exact solution to the NRP problem, i.e., a partition allocation $P_i^{\mathcal{R}}$ for each application $\mathcal{A}_i$ on resource $\mathcal{R}$. Then, each application can generate a feasible schedule using the allocated partition to satisfy the timing requirements of all the tasks.

VI. Supply Graph based Partitioning

Since the SMT-based exact solution does not scale with the network and application size, in this section we propose an efficient supply graph based partitioning framework, SGP, to solve the NRP problem.

Note that, [24] proposes a resource partitioning algorithm AAF which can be used to solve the NRP problem. However, AAF suffers from the limitation that the regularity of resource interface must be an integer value. This limitation significantly degrades the performance of AAF which is revealed in our experimental results in Section VII. Furthermore, the partitioning algorithms proposed for the RRP model do not consider the unique challenge posed by the node dependency in RTWNs. Therefore, the proposed supply graph based partitioning framework SGP aims to break the regularity limitation to support any non-integer regularity values, and handle node dependency among different applications, to improve the resource partitioning performance.

A. Overview

The high-level idea of SGP is to determine the resource partition for all the applications in a slot-by-slot fashion within $[1, T]$, i.e., the slotframe length, in two rounds. In the first round, we perform channel resource partitioning for all the applications to satisfy their channel resource interface requirements. In the second round, based on the channel partition allocated to each application, we perform critical node resource partitioning to satisfy the node resource interfaces and the node dependency requirement for each application.

The resource partitioning in each time slot is performed according to the *weight* of each application on the requirement of a particular resource. A higher weight indicates that the application urges the allocation of the resource, otherwise the corresponding resource interface may not be satisfied. Then, in each time slot, the network manager allocates each resource (either channel or a critical node) to the application with the highest weight on the request of this resource.

Then, the question is how to determine the weight of each application to capture the resource requirement urgency. According to the discussions in Section IV-B and Section IV-C, the requirement urgency of each application in a time slot is determined from two aspects: i) the resource interface (i.e., the availability factor α and regularity r), and ii) the node dependency if the resource is a critical node (i.e., satisfying the node precedence graph $\mathcal{G}_i^P$ and node conflict graph $\mathcal{G}_i^C$).

To derive the weight of each application $\mathcal{A}_i$ on a particular resource $\mathcal{R}$ in each time slot t, denoted as $w_i^{\mathcal{R}(t)}$, from the above two aspects, we i) leverage a *supply graph* to assess how urgently application $\mathcal{A}_i$ requires resource $\mathcal{R}$ in slot t, and ii) develop a set of rules to decide the weight according to the node dependency graphs. Algorithm 1 presents the main functions of SGP. Below, we describe how to determine weight $w_i^{\mathcal{R}(t)}$ from each aspect.

B. Supply Graph

In this section, we describe how to use the supply graph to determine the weight of each application $\mathcal{A}_i$ on resource $\mathcal{R}$ in slot t according to $\mathcal{A}_i$'s resource interface $I_i^{\mathcal{R}}$. Essentially, the resource interface $I_i^{\mathcal{R}}$ captures i) the amount of resource $\mathcal{R}$ required by $\mathcal{A}_i$ in each slotframe (specified by availability factor α), and ii) the distribution of $\mathcal{R}$ in each slotframe (specified by regularity r).

Therefore, we leverage a supply graph to describe i) the amount of resource $\mathcal{R}$ that is already allocated to $\mathcal{A}_i$, and ii)

Algorithm 1 Supply Graph based Partitioning

for $t = 1$ to T **do**
  $W \leftarrow$ all applications' weight $w_i^{C_i(t)}$ on C_i channels;
  Sort W in the descending order;
  for $w_i^{C_i}(t) \in W$ **do**
    if Number of remaining channels $\geq C_i$ **then**
      Append t to partition $P_i^{C_i}$;
for $t = 1$ to T **do**
  for $v^* \in$ critical nodes set **do**
    Collect all applications' weight $w_i^{v^*(t)}$ on v^*;
    Allocate v^* to the application $\mathcal{A}_i$ with highest weight;
    Append t to partition $P_i^{v^*}$;
return partitions

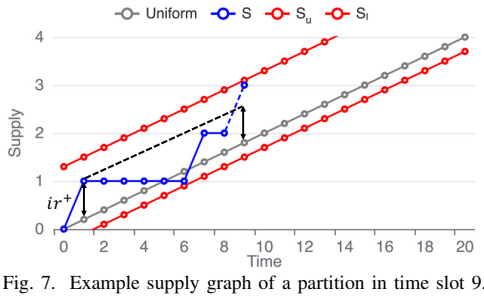


Fig. 7. Example supply graph of a partition in time slot 9.

the requirement of $\mathcal{R}$ specified by the resource interface. Based on these two information, we are able to determine whether $\mathcal{R}$ is required by $\mathcal{A}_i$ in the current slot and how urgent is the requirement, i.e., the weight value $w_i^{\mathcal{R}(t)}$. The supply graph of $\mathcal{A}_i$ consists of three functions.

Supply function $S(P,t)$ denotes the number of slots allocated to P during the time interval $[0,t]$.

Supply upper bound function $S_u(P,t) = ir^-(P,t) + U(P,t) + r$, where $ir^-(P,t) \leq 0$ is the minimum instant regularity within $[0,t]$.

Supply lower bound function $S_l(P,t) = ir^+(P,t) + U(P,t) - r$, where $ir^+(P,t) \geq 0$ is the maximum instant regularity within $[0,t]$.

If the supply function falls within the range of the upper and lower bound functions, i.e., $S_l(P,t) < S(P,t) < S_u(P,t)$, the partition satisfies the regularity constraint. Thus, the supply upper and lower bound functions tell us whether resource $\mathcal{R}$ ***can*** be allocated to application $\mathcal{A}_i$ in the current slot t to satisfy the regularity requirement. Below, we introduce a lemma which tells us if resource $\mathcal{R}$ can be allocated to $\mathcal{A}_i$ in slot t, whether $\mathcal{R}$ ***should*** be allocated to $\mathcal{A}_i$.

Lemma 1. *If the instant regularity generated by allocating $\mathcal{R}$ to $\mathcal{A}_i$ in the current slot t is larger than $ir^+(P,t-1)$ (less than $ir^-(P,t-1)$), the supply lower (upper) bound function is increased (reduced).*

The proof of Lemma 1 is straightforward according to the definitions of supply upper and lower bound functions, and thus is omitted here due to the page limit. Lemma 1 indicates that keeping the instant regularity within a small range will not shrink the feasible range of $S(P,t)$ specified by $(S_l(P,t), S_u(P,t))$. That is, the number of feasible time slots for the next allocation of resource $\mathcal{R}$ to application $\mathcal{A}_i$ is not reduced. This is important since the network manager needs to accommodate multiple applications to satisfy their individual resource interfaces. If the feasible solution space for the resource allocation of each application is reduced, it is more difficult for the network manager to find a feasible solution for the resource allocation of all the applications.

Based on the above discussions, weight $w_i^{\mathcal{R}(t)}$ of each application $\mathcal{A}_i$ on resource $\mathcal{R}$ in slot t can be determined based on the following three cases.

Case 1. If allocating $\mathcal{R}$ to $\mathcal{A}_i$ in the current slot t causes $S(P,t)$ to exceed the range of $(S_l(P,t), S_u(P,t))$, we set $w_i^{\mathcal{R}(t)} = 0$. That is, $\mathcal{R}$ cannot be allocated to $\mathcal{A}_i$, otherwise the regularity requirement specified in the resource interface requirement cannot be met.

Case 2. In this case, allocating $\mathcal{R}$ to $\mathcal{A}_i$ in the current slot t does not cause $S(P,t)$ to exceed the range of $(S_l(P,t), S_u(P,t))$, but the instant regularity exceeds the range of $(ir^-(P,t-1), ir^+(P,t-1))$. Then, we set $w_i^{\mathcal{R}(t)} = 1/\max(t_e - t, 1)$, where t_e is the latest time slot that $\mathcal{R}$ must be allocated to $\mathcal{A}_i$ to satisfy the regularity requirement. That is, t_e is the deadline to allocate $\mathcal{R}$ to $\mathcal{A}_i$ and is used as the metric to capture the urgency on the requirement of $\mathcal{R}$.

Case 3. If allocating $\mathcal{R}$ to $\mathcal{A}_i$ in the current slot t does not cause the instant regularity to exceed the range of $(ir^-(P,t-1), ir^+(P,t-1))$, we set $w_i^{\mathcal{R}(t)} = 1/\max(t_e - t, 1) + 1$. That is, according to Lemma 1, an additional weight value is added in this case that the instant regularity is within a small range.

Fig .7 shows an example of a supply graph of an application in time slot 9, where the blue curve represents supply function $S(P,t)$ and the red curves represent the supply upper and lower bound functions. Then, if the resource is allocated to the application in slot 9, we have $S(P,9) < S_u$ but the instant regularity $ir(P,9) > ir^+$ (i.e., Case 2). Thus, the weight of the application in slot 9 is $\frac{1}{\max(12-9,1)} = 1/3$ since $t_e = 12$.

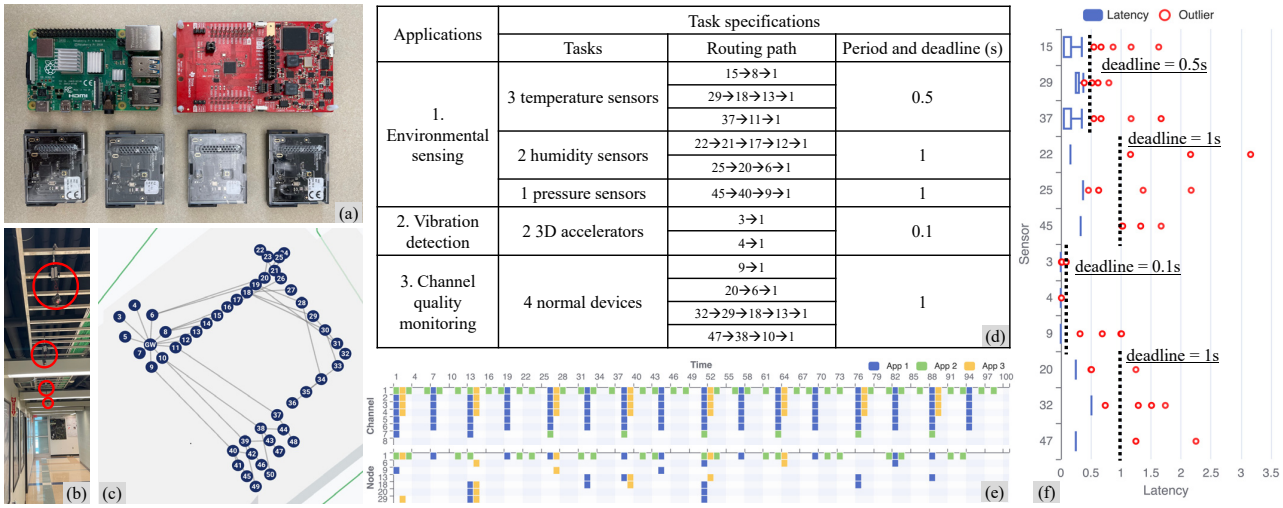
C. Satisfying Node Dependency

If resource $\mathcal{R}$ is a critical node, in addition to the resource interface requirement captured by the supply graph, we need to consider the node dependency constraints when we determine the weight $w_i^{\mathcal{R}(t)}$ for each application.

According to Section IV-C, intra-application node dependency is captured by the node precedence graph and node conflict graph. Thus, based on the weight generated according to the resource interface of each critical node, we develop a set of rules to refine the weight, according to the constraints posed by each dependency graph.

1) Rules for the precedence graph: The constraints specified by the node precedence graph restrict the access order of the critical nodes on the routing path of a same task. Thus, we develop the following two rules to refine the weight of each critical node.

Rule 1. For any critical node v_j, If any node in the predecessor vertices of v_j in the node precedence graph $\mathcal{G}_i^P$ does not



Applications	Task specifications		
	Tasks	Routing path	Period and deadline (s)
1. Environmental sensing	3 temperature sensors	15→8→1	0.5
		29→18→13→1	
		37→11→1	
	2 humidity sensors	22→21→17→12→1	1
		25→20→6→1	
	1 pressure sensors	45→40→9→1	1
2. Vibration detection	2 3D accelerators	3→1	0.1
		4→1	
3. Channel quality monitoring	4 normal devices	9→1	1
		20→6→1	
		32→29→18→13→1	
		47→38→10→1	

Fig. 8. 6TiSCH testbed experiments. (a) Hardware for the gateway (RPi4B and CC2652) and device nodes (SensorTag); (b) deployment in the building; (c) network topology on google map; (d) applications deployed on the testbed; (e) channel and node partitions; (f) end-to-end latency result of each source node.

receive the required number of slots, the weight for node v_j is set to 0.

Rule 2. For two nodes v_j, v_k in the same vertex in $\mathcal{G}_i^P$, their weight is set to the greater of their original weights, i.e., $w = \max(w_i^{v_j}, w_i^{v_k})$, to ensure the application can access these two nodes simultaneously.

2) Rules for the conflict graph: The constraints specified by the node conflict graph avoid the simultaneous transmissions from any two nodes with a same sender/receiver. Thus, we develop the following

Rule 3. For any two nodes connected in the node conflict graph, the weight of one node is set to 0.

Note that, since each node may be connected to multiple nodes, we follow a greedy manner to determine which nodes' weights are set to 0 in a conflict graph. Specifically, we select the node with the largest weight and set the weight(s) of the connected node(s) to 0. This process repeats until the weights of all the nodes in the conflict graph are refined.

3) Other rules: Since one transmission can be transmitted only if both channel resource and corresponding node resources are available. Thus, we have the following rule.

Rule 4. If the application is not allocated with channel resource in the current slot, the weights of all the critical nodes are set to 0.

VII. Performance Evaluation

In this section, we present the experimental results of the proposed network resource partitioning framework through i) testbed validation, ii) MAC layer emulator, and iii) synthetic data set.

A. 6TiSCH Testbed Validation

We implement the network resource partitioning framework on a 5-hop 8-channel 6TiSCH network testbed with 49 devices. 6TiSCH is a representative multi-hop multi-channel RTWN technology. It integrates the IEEE 802.15.4e TSCH data link layer with an IP-enabled upper stack (using 6LoWPAN). to achieve both deterministic real-time performance with ultra-low power consumption and seamless integration with Internet services. The network layer of 6TiSCH uses RPL routing protocol [29] to form tree topology. We use TI CC2650 SensorTag as the device nodes and a RaspberryPi 4B device attached with a TI CC2652 board to serve as the gateway running the network manager. Each SensorTag is equipped with multiple sensors, e.g., temperature, pressure, humidity, gyro and accelerators. The 6TiSCH full stack is implemented on TI-RTOS [30] and deployed on both device nodes and the gateway. Fig. 8(a) and Fig. 8(b) show the hardware used in the testbed and their deployment in the labs and hallway. Fig. 8(c) shows the network topology and GPS location of the devices. The duration of each time slot is 10 ms and the slotframe length is set to $T = 100$ (1 second) in the experiments. 8 of the IEEE 802.15.4e channels are enabled for communications, i.e., $C = 8$.

We deploy three applications on the 6TiSCH testbed and the task specifications are shown in Fig. 8(d). Fig. 8(e) shows the channel and node partitions allocated to each application for local schedule generation under EDF scheduling for its tasks. We run the network for 30 minutes and collect the end-to-end latency results from each source node (Fig. 8(f)). The results show that with the schedule constructed on the allocated partitions, almost all packets meet their deadlines. However, due to the high interference from other wireless devices in the building, packet loss occurs and causes retransmissions, which increase the end-to-end latency.

B. RTWN Emulator

In this set of experiments, we develop an RTWN emulator to emulate large-scale mesh networks (see Fig. 9 with a screenshot) and generate realistic network task sets to compare the schedulability of SGP to that of other methods. We implement the RTWN virtualization framework which consists

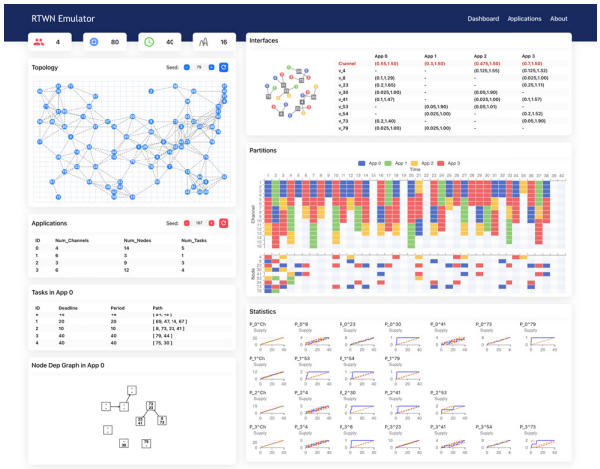


Fig. 9. A screenshot of the high-fidelity RTWN emulator. Key components including nodes, application, manager and network statistics visualization.

of the network nodes, applications and network manager. In each experiment, we first randomly distribute $|V|$ network nodes on a grid with a fixed size (32×24). Nodes within each other's transmission range (setting to 6) can establish a wireless link and eventually all the nodes form a multi-hop mesh network. We then generate N applications, each with a scheduling policy randomly selected from multi-channel EDF and RM policies and a randomly selected number of available channels C_i. For each application, we generate a task set with up to M tasks. Each application submits the channel/node resource interfaces to the network manager, and constructs local schedule for its tasks upon receiving the resource partitions from the network manager.

In this set of experiments, the metric used is **Task Success Ratio (TSR)**, which is defined as the ratio of application sets satisfying tasks' timing requirements to all the generated application sets. Fig. 10(a-c) illustrate the TSR comparisons among different algorithms and partitioning models, including SGP and SMT approach, as well as the RRP partition model, EDP partition model [17], and a Round-Robin (RR)-based partition allocation approach. Each data point represents the average value obtained from 500 trials.

As the number of applications N or tasks M increases, there is a higher degree of resource competition within the network, resulting in a decrease in the TSR for all the methods. On the other hand, a higher value of $|V|$ indicates an increased network capacity, leading to a higher TSR. The results demonstrate that SGP significantly outperforms other algorithms and achieves a TSR comparable to that of the SMT approach. The classic RRP model, however, is unable to handle node dependency and exhibits a much lower TSR. The EDP partition model primarily focuses on meeting throughput requirements by allocating a certain amount of resources within a specified period. However, when a task set comprises tasks with diverse periods and deadlines, its compositional approach fails to ensure the task set schedulability. RR achieves near-uniform resource allocation and can satisfy the timing requirements, but it only performs well under light network workload settings.

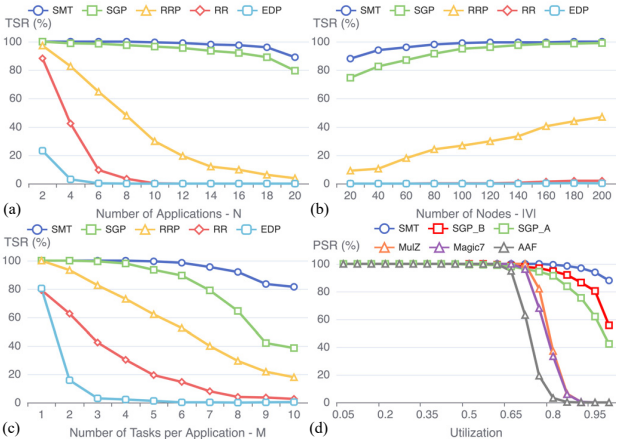


Fig. 10. (a-c) TSR comparison results on RTWN emulator: (a) set $|V| = 120, M = 3$ and vary N; (b) set $N = 10, M = 3$ and vary $|V|$; (c) set $N = 4, |V| = 120$ and vary M. (d) PSR comparison results on synthetic data set.

C. Synthetic Data Set

We compare the partition allocation performance of SGP with SMT solution and the RRP partitioning algorithms: AAF [24], Magic7 [25], MulZ [31] with synthetic resource interfaces for one single resource. Two variations of SGP are included: SGP_A that only considers Case 1 and Case 2 in determining the weight, and SGP_B uses Lemma 1 and Case 3. The performance metric used is **Partition Success Ratio (PSR)**, which is defined as the ratio of application sets received feasible partitions satisfying interfaces to all the generated applications sets.

We generate 20 applications, each of which has a resource interface. The slotframe length is set to 112. The total utilization (sum of availability factors of each interface) ranges from 0 to 1 and the regularity of each partition ranges from 1 to 2. For each utilization setting, we repeat 2000 trials and collect the average result. Fig. 10(d) shows the PSR of all the methods decrease with the increasing of utilization, and SMT dominates others as an exact solution. The PSR gap between SMT and SGP_B is small and validates the effectiveness of SGP. On the other hand, RRP-based algorithms degrade significantly due to their large approximation overhead and lack of support for non-integer regularity values.

VIII. Conclusion and Future Work

This paper formulates the network resource partitioning problem for RTWNs to support multiple applications with end-to-end timing requirements. We present a SMT-based exact solution and a supply graph based resource allocation framework to further reduce the computation overhead. The effectiveness of the proposed solutions are demonstrated through extensive testbed experiments and performance evaluation.

As for the future work, we will extend the proposed supply graph based partitioning framework to support lossy links . Another important direction is to explore how to reconfigure the resource allocations at run time with minimum overhead in the presence of unexpected network dynamics.

IX. Acknowledgement

The work is supported in part by the National Science Foundation (NSF) Grant CNS-1932480, CNS-2008463 and CCF-2028875.

References

[1] J. Song *et al.*, "WirelessHART: Applying wireless technology in real-time industrial process control," in *RTAS*. IEEE, 2008, pp. 377–386.

[2] The International Society of Automation, "ISA 100.11a." [Online]. Available: https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa100

[3] D. Dujovne *et al.*, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, 2014.

[4] Z. Yun *et al.*, "RT-WiFi on software-defined radio: Design and implementation," in *IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 254–266.

[5] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1013–1024, 2015.

[6] 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 *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 261–272.

[7] T. Zhang, T. Gong, Z. Yun, S. Han, Q. Deng, and X. S. Hu, "Fd-pas: A fully distributed packet scheduling framework for handling disturbances in real-time wireless networks," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 1–12.

[8] T. Gong, T. Zhang, X. S. Hu, Q. Deng, M. Lemmon, and S. Han, "Reliable dynamic packet scheduling over lossy real-time wireless networks," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[9] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE transactions on industrial informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[10] D. Shen, T. Zhang, J. Wang, Q. Deng, S. Han, and X. S. Hu, "Distributed successive packet scheduling for multi-channel real-time wireless networks," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2022, pp. 71–80.

[11] T. Zhang, T. Gong, X. S. Hu, Q. Deng, and S. Han, "Dynamic resource management in real-time wireless networks," *Wireless Networks and Industrial IoT: Applications, Challenges and Enablers*, pp. 131–156, 2021.

[12] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 358–380, 2014.

[13] I. Afolabi *et al.*, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[14] J. Wang, T. Zhang, D. Shen, X. S. Hu, and S. Han, "Apas: An adaptive partition-based scheduling framework for 6tisch networks," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 320–332.

[15] W.-J. Chen *et al.*, "Online reconfiguration of regularity-based resource partitions in cyber-physical systems," *Real-Time Systems*, vol. 57, no. 3, pp. 302–345, 2021.

[16] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS*. IEEE, 2003, pp. 2–13.

[17] A. Easwaran *et al.*, "Compositional analysis framework using edp resource models," in *RTSS*. IEEE, 2007, pp. 129–138.

[18] T. Amert *et al.*, "Timewall: Enabling time partitioning for real-time multicore+ accelerator platforms," in *RTSS*. IEEE, 2021, pp. 455–468.

[19] S. Voronov, S. Tang, T. Amert, and J. H. Anderson, "Ai meets real-time: Addressing real-world complexities in graph response-time analysis," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 82–96.

[20] G. Dai, P. K. Paluri, and A. M. Cheng, "Enhanced schedulability tests for real-time regularity-based virtualized systems with dependent and self-suspension tasks," *Real-Time Systems*, pp. 1–29, 2022.

[21] J. Wang, T. Zhang, D. Shen, X. S. Hu, and S. Han, "Harp: Hierarchical resource partitioning in dynamic industrial wireless networks," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2022, pp. 1029–1039.

[22] C. Wu *et al.*, "Analysis of edf scheduling for wireless sensor-actuator networks," in *IWQoS*. IEEE, 2014, pp. 31–40.

[23] S. Ghosh, S. Dey, and P. Dasgupta, "Pattern guided integrated scheduling and routing in multi-hop control networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 19, no. 2, pp. 1–28, 2020.

[24] A. K. Mok and X. Alex, "Towards compositionality in real-time resource partitioning based on regularity bounds," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE, 2001, pp. 129–138.

[25] Y. Li and A. M. Cheng, "Static approximation algorithms for regularity-based resource partitioning," in *2012 IEEE 33rd Real-Time Systems Symposium*. IEEE, 2012, pp. 137–148.

[26] Y. Li and A. M. K. Cheng, "Transparent real-time task scheduling on temporal resource partitions," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1646–1655, 2015.

[27] Q. He, N. Guan, Z. Guo *et al.*, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2283–2295, 2019.

[28] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 128–140.

[29] "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," https://datatracker.ietf.org/doc/html/rfc6550.

[30] Texas Instruments, *TI-RTOS 2.20 User's Guide*, 2016. [Online]. Available: http://www.ti.com/lit/ug/spruhd4m/spruhd4m.pdf

[31] G. Dai, P. K. Paluri, A. M. K. Cheng, and B. Liu, "Regularity-based virtualization under the arinc 653 standard for embedded systems," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2592–2605, 2021.