2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00103

# HARP: Hierarchical Resource Partitioning in Dynamic Industrial Wireless Networks

Jiachen Wang†*, Tianyu Zhang†, Dawei Shen§, 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
§School of Computer Science and Engineering, Northeastern University, Shenyang, China
§Email: 1610547@stu.neu.edu.cn
¶Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556
¶Email: shu@nd.edu

***Abstract*—Industrial wireless networks (IWNs) are being increasingly deployed in the field to serve as the network fabrics for various industrial Internet-of-Things (IIoT) applications. Given that IWNs typically operate in noisy and harsh environments, frequently occurring network dynamics post huge challenges for IWN resource management especially when the network scales up. Existing centralized and distributed network management solutions either suffer from large communication overhead and time delay, or introduce schedule collisions which unnecessarily degrade the system performance. To address these problems, this work proposes a novel HierArchical Resource Partitioning framework (HARP), to provide dynamic resource management in IWNs. By hierarchically partitioning and allocating resources for the links in the network, HARP enables distributed collision-free resource allocation. HARP enables rapid adjustment of the partitions in the presence of network dynamics with modest communication overhead. The effectiveness of HARP is validated and evaluated through both simulation studies and testbed experiments on a 50-node multi-channel multi-hop 6TiSCH network.**

***Index Terms*—Industrial wireless networks, hierarchical resource partitioning and reconfiguration, network dynamics**

## I. Introduction

Industrial Internet-of-Things (IIoT) systems are gaining rapid adoption in process control industries including but not limited to chemical process control, automotive and aerospace manufacturing. As a key component of the IIoT network fabric, Industrial Wireless Networks (IWNs) are typically deployed in noisy and harsh environments where interference and disturbance may occur throughout the network lifetime. Interference can cause the network nodes to change their connected nodes to seek for more reliable links, which changes the network topology. Unexpected disturbance often requires the system to either modify existing or activate new network functions to react to unexpected events, leading to traffic changes. To ensure that packet transmissions meet stringent real-time requirements in IIoT applications, IWN resource management, i.e., determining the communication schedule (which packet transmission should use which time slot and which channel), is indispensable. The network dynamics discussed above unavoidably introduce instability to the system operation and make an originally feasible communication schedule no longer acceptable. Thus, it is necessary to reconfigure the communication schedules to adapt to the updated network topology and resource requirements in a timely fashion.

IWNs typically adopt centralized network resource management where a single node (e.g., gateway) maintains the global information of the entire network and decides the resource allocation for all the links. A number of centralized scheduling approaches for handling network dynamics have been proposed (e.g., [1]–[6]). Under centralized resource management, when network dynamics happen, traffic change requests are sent from the affected node(s) to the gateway through predetermined routing path(s). The gateway then constructs an updated communication schedule to achieve an optimal reconfiguration based on the global network information. Upon receiving the updated configuration information from the gateway, each individual node deploys the updated schedule to accommodate the network traffic change. Centralized network management, however, suffers from both large communication overhead and significant time delay, especially when the network scales up, since multi-hop network management packets are required to communicate the updated scheduling information. Frequent traffic and network topology changes in harsh industrial environment further aggravate this problem and hinder centralized network resource management from being applied in practice.

Many research efforts (e.g., [7]–[9]) have been devoted on designing distributed managements to overcome the aforementioned drawbacks of centralized managements for more flexible and faster resource allocation and reconfiguration in the presence of network changes. [7] proposes a distributed traffic-aware scheduling algorithm for IEEE 802.15.4e networks where the traffic requirements of individual nodes are collected in a distributed fashion in runtime. To reduce communication overhead and response time, [8] proposes a distributed dynamic packet scheduling framework, FD-PaS, for IWNs to handle network disturbances. However, fully distributed network management faces severe challenges in avoiding communication schedule conflicts among individual nodes. This is caused by the fact that each node only maintains a portion of the entire network information (e.g., link quality, task specification) and their communication schedules are constructed in a local fashion.

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

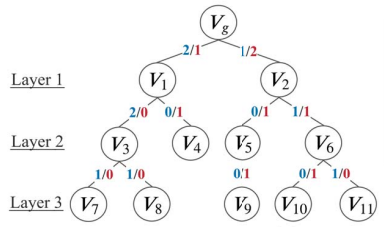
2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00103

Several works in the literature studied the transmission collision elimination problem in IWNs. The distributed scheduler MSF [10] defined in the 6TiSCH standard performs schedule reconfiguration once transmission collisions are detected through monitoring the packet delivery ratio (PDR) of each cell. [9] proposes to share the schedule information among the one-hop neighbor nodes in the broadcasting phase to reduce transmission collisions. [11] relies on the shared slots in the slotframe to overhear the schedules distributedly generated by the neighbor nodes in the network bootstrap phase. ALICE [12] is proposed as a hash-based distributed scheduler, and it allows the sender and receiver nodes to generate a new schedule in each slotframe. DistributedHART [13] applies the distributed vertex coloring approach to generate conflict-free schedules locally based on the interference information stored by each node. However, all the aforementioned approaches cannot *guarantee* collision-free transmissions during the network operation and may introduce unnecessary communication overheads to generate consistent schedules among different nodes. Achieving collision-free communications among nodes in the network is vital to guarantee the performance of IWNs.

From the above discussion, we can observe that neither centralized nor fully distributed network resource management in IWNs can handle network changes effectively. To address this problem, we propose HARP, a novel HierArchical Resource Partitioning framework for dynamic management of network resources in IWNs. HARP models the routing topology of a IWN as a tree structure which is commonly deployed for industrial control applications[1]. By hierarchically partitioning and allocating resource for different subtrees in the network, HARP provides sufficient and dedicated resource for each link, and thus provides distributed collision-free resource allocation. Based on the hierarchical resource management strategy, HARP can promptly adapt to different traffic scenarios by adjusting only a limited number of nodes and reconfiguration messages. The main contributions of this work are as follows:

1) We introduce a novel hierarchical resource partitioning based resource management framework for IWNs to allocate dedicated resource for each link to support collision-free distributed scheduling.
2) We design abstraction and composition methods to capture the resource requirements for IWNs in a hierarchical fashion. We also design an effective resource negotiation process to allocate resource partitions based on the proposed resource requirement model.
3) We propose an effective partition adjustment method to create new or reconfigure existing partitions in the network to adapt the resource partitions to topology and traffic changes during runtime.
4) We implement the proposed HARP framework on a 5-hop 50-node 6TiSCH network [14] (a typical IWN), and validate HARP's correctness and effectiveness through both testbed experiments and simulation studies.

[1]For non-tree based routing topology, one could decompose the topology to multiple tree structures and apply HARP in a divide and conquer fashion. We leave the details of this to future work.




| Task | Total number of cells |
|---|---|
| $V_7 \rightarrow V_{10}$ | 6 |
| $V_8 \rightarrow V_9$ | 6 |
| $V_{11} \rightarrow V_4$ | 5 |

(a)

time slots

| channels | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $e_{7,3}$ | $e_{3,1}$ | $e_{1,g}$ | $e_{g,2}$ | $e_{2,6}$ | $e_{6,10}$ | | | | |
| 2 | $e_{11,6}$ | $e_{6,2}$ | | | | | | $e_{2,g}$ | $e_{g,1}$ | $e_{1,4}$ |
| 3 | | | $e_{8,3}$ | $e_{3,1}$ | $e_{1,g}$ | $e_{g,2}$ | $e_{2,5}$ | $e_{5,9}$ | | |
| 4 | | | | | | | | | | |

(b)

Fig. 1. (a) An IWN topology with 12 nodes and 3 layers. There are 3 tasks in this network. The number of cells required by each uplink and downlink are shown in the left and right of the slash, respectively. (b) An example schedule accommodates the task set in this network.

## II. Network Model and Problem Description

In this section, we present the IWN model and describe the problem to be studied in this work.

### A. Network Model

We adopt a typical multi-channel multi-hop IWN model, in which a set of sensors and actuators are wirelessly connected to a gateway either directly or through one or multiple relay nodes. Each node in the IWN is equipped with a single omni-directional antenna operating in the half-duplex mode. To simplify the HARP design, we model the network topology as a tree structure which is commonly adopted in representative IWNs such as 6TiSCH and ZigBee [15]. We denote a network tree topology as $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where the node set $\mathbf{V} = \{\{V_1, V_2, \cdots, \}, V_g\}$ correspond to sensor, actuator and relay nodes, and the root node $V_g$ represents the gateway. Given the tree topology, each node only has one parent node but can have multiple child nodes. Link $e_{i,j} \in \mathbf{E}$ represents the directed wireless communication between nodes $V_i$ and $V_j$, where $V_i$ is the sender and $V_j$ is the receiver. Each link is associated with an attribute *layer* ($l$) which equals the child node's hop count to the gateway.

A subtree with $V_i$ as the root node is denoted by $\mathbf{G}_{V_i}$. If $\mathbf{G}_{V_i} \subset \mathbf{G}_{V_j}$, we say $\mathbf{G}_{V_i}$ is a subtree of $\mathbf{G}_{V_j}$. If $V_i$ is a child node of $V_j$, we say $\mathbf{G}_{V_i}$ is a *direct subtree* of $\mathbf{G}_{V_j}$. The links connecting $V_i$ and its children all have the same layer value, thus we use $l(V_i)$ to denote their layers. We define the layer of subtree $\mathbf{G}_{V_i}$, denoted by $l(\mathbf{G}_{V_i})$, as the largest layer value among all the links in $\mathbf{G}_{V_i}$. Fig. 1(a) depicts an example 12-node IWN with the layer values for different links.

In our network model, we employ a multi-channel Time Division Multiple Access (TDMA) based data link layer which is the most common setting for IWNs (e.g., WirelessHART [16], ISA100.11a [17] and 6TiSCH), and adopts

a link-based scheduler to allocate the network resource to individual links. We use the concept of *cell* to represent the basic unit of the network resource that can be allocated to the links and a cell is denoted by a tuple (slot, channel). Consecutive time slots are grouped into *slotframes* and the assignment of cells to individual links in a slotframe defines the network schedule which repeats every slotframe during the network operation.

We use the concept of *task* to represent the data flow from sensor(s) to actuator(s). Following real-life IWN settings, we assume that each task *periodically* samples the designated physical entity and sends the sensor readings along a pre-defined uplink routing path to the gateway for data collection and control decision-making. The generated control signals at the gateway are then forwarded along a pre-defined downlink routing path to an actuator for execution. The information transmitted for one instance of a task is referred to as a packet.

In link-based scheduling, each cell is allocated to a link by specifying the sender and receiver to transmit a packet through the link. Task-level resource requirements can be abstracted to link-level cell requirements by an existing method (e.g., [4], [18]). Thus, in this work we assume that the number of cells required by each link $e_{i,j}$, denoted by $r(e_{i,j})$, is provided based on the tasks' routing paths to satisfy their requirements, and each node only maintains the cell requirements for the links passing through it. Fig. 1(b) depicts an example schedule constructed based on an example task set of three tasks. The corresponding link-level cell requirements of the tasks are shown on the right side of Fig. 1(a).

### B. Problem Description

The objective of this work is to design a network resource management framework for IWNs to handle network dynamics in an efficient and effective manner. The goal of the management framework is to determine the cells usable by each link $e_{i,j}$ for data transmission, i.e. the network schedule. To avoid collision, a cell should not be assigned to more than one link and the total number of cells allocated to link should be no smaller than the link's required number of cells.

To achieve efficiency, the network management framework should be able to respond to the network dynamics in a timely manner without introducing unnecessary communication overhead. As an effective framework, it should guarantee that no schedule collision occurs among all the links even in the process of handling network changes. As we have discussed earlier, centralized management approaches cannot offer the desired efficiency while fully decentralized management approaches often lead to ineffective assignments since it cannot avoid transmission collisions due to the lack of the global information at individual nodes. In the following sections, we describe the design of our resource management framework (HARP) in detail and show how it can satisfy the efficiency and effectiveness requirements.

## III. Overview of the HARP Framework

We first present an overview of the HARP framework for managing network resources in IWNs, and adapt to network dynamics during runtime. The key idea of HARP is to divide a slotframe into a hierarchy of partitions and assign the partitions to specific links of each subtree based on those links' layer information. Each partition is associated with a node index and layer index. Such a layered partition design, i.e., allocating one partition for a layer of a subtree, improves the efficiency of cell usage and we will describe the details using a motivational example in the next section.

The above hierarchical, subtree-based resource management has two advantages. First, it can achieve network resource isolation for those links in different subtrees and thus avoid transmission collisions. Second, network dynamics can be handled by only adjusting the cell assignment for links in the relevant subtree instead of the entire network, which significantly reduces the network management overhead.

Fig. 2 describes the execution model of HARP. Overall, HARP consists of three phases after the network bootstrapping. In the *static partition allocation phase*, partitions are first created for individual subtrees and then cells are allocated for individual links in the corresponding partitions based on their resource requirements. Specifically, the resource requirements of all the links are abstracted and composed in a bottom-up fashion from the subtrees at the lowest level of the network to the gateway. Based on those requirements, resource partitions are then allocated in a top-down fashion from the gateway to the root node of each subtree. After that, in the *distributed scheduling phase*, the root node of each subtree constructs the communication schedule (i.e., cell assignment) within its allocated resource partition for all the links between itself and its child nodes. During the network operation, upon any traffic change being requested by a certain node $V_i$, the *dynamic partition adjustment phase* is triggered. If idle cells are available within the partition allocated to the subtree with $V_i$ as the root, $V_i$ directly assigns more cells within the partition to accommodate the increased traffic. Otherwise, a partition adjustment request will be sent to the parent of $V_i$. This process repeats until the partition of a node is able to satisfy the traffic change request. Then the updated partition information is propagated downward to $V_i$.

The details of the three phases of HARP will be presented in the next two sections.

## IV. Static Partition Allocation

In this section, we describe the static partition allocation phase of HARP. To achieve efficient resource allocation, we need to first determine how to efficiently capture the resource requirement of each subtree. Note that, in the following we do not distinguish between uplink and downlink since they are handled in the same way in HARP.

### A. Resource Requirement Representation Design

The representation of a subtree's resource requirement should possess the following two desirable properties. First,

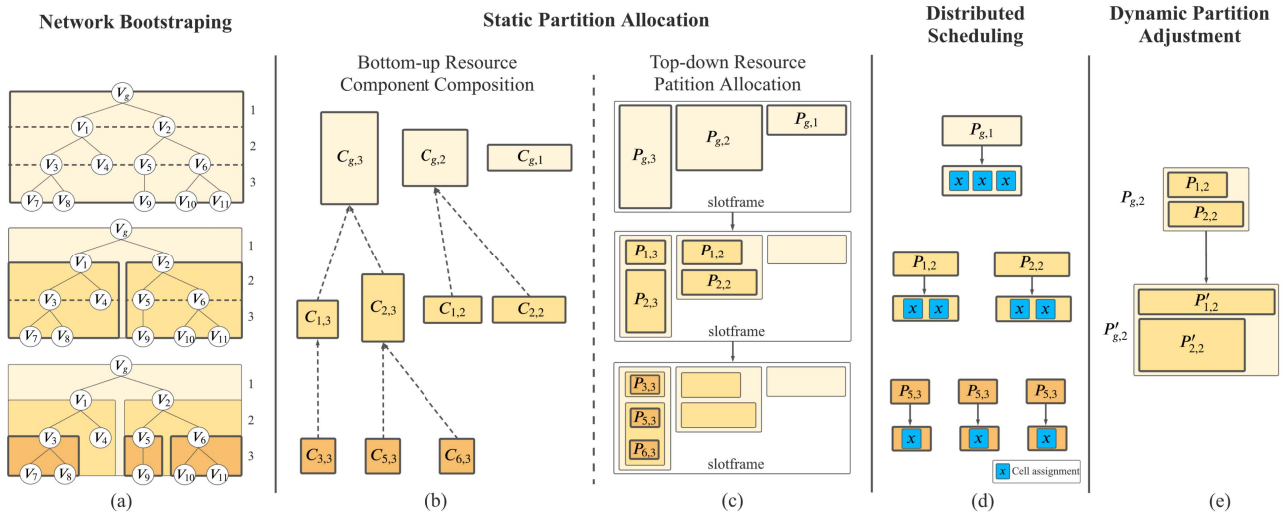



Fig. 2. Overview of the HARP execution model. (a) Network topology with all the subtrees highlighted in different colors. (b) Resource components generated for different subtrees. (c) Partitions allocated to subtrees. (d) Distributed cell assignments determined by the root node of all the subtrees. (e) An example dynamic partition adjustment.

it should capture the requirements of all the links in the subtree correctly. Although the resource requirement (i.e., the number of cells) needed by each link is assumed to be given, we cannot simply accumulate the number of cells. This is because each node operates in a half-duplex mode and links sharing a common node cannot be assigned to cells within a same time slot. Thus, the resource requirement of a subtree needs to not only specify the number of cells but also the number of time slots and channels satisfying the half-duplex constraint. Second, the data structure should be compact to reduce the storage and communication overhead, given that this information needs to be propagated to the gateway through multiple nodes at multiple layers. Keeping these two properties in mind, we use number of time slots and number of channels to capture the resource requirement of a subtree. Pictorially this can be represented as a *rectangular* resource block, where the length of the rectangle indicates the number of time slots and the height of the rectangle indicates the number of channels.

According to the analysis in [19], a routing-path compliant schedule (i.e., the sequence of the cells allocated to the links of a packet at different layers follows the packet's routing path) can significantly reduce the end-to-end latency of the packet. Therefore, the cells allocated to all the links within a subtree in one rectangular resource block should follow the order of the packet's routing path. In this case, using one rectangle block to abstract the resource requirements of all the links within a subtree can cause resource waste and reduce the cell usage. For example, consider two subtrees $\mathbf{G}_{V_i}$ and $\mathbf{G}_{V_j}$ with $V_i$ and $V_j$ as root nodes, respectively, and each subtree contains links located at two layers. As shown in Fig. 3(a), resource allocated to the links at different layers is separated by a dashed line in the corresponding rectangular resource block for $\mathbf{G}_{V_i}$ and $\mathbf{G}_{V_j}$. We can observe that the resource in the white area within each rectangle is wasted. To improve

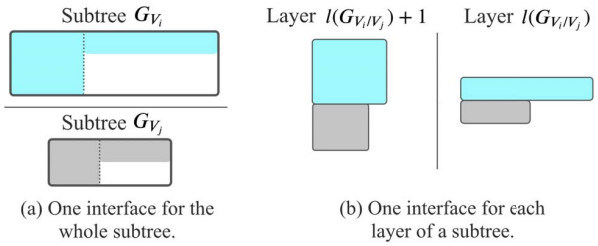



Fig. 3. An motivating example for layered interface design.

the cell usage, we propose a layered resource interface design to capture the resource requirement of a subtree at each layer separately (as shown in Fig. 3(b)). Then, the resource interface of a subtree contains a set of rectangular resource blocks each of which captures the resource requirements of links at a particular layer.

Below, we give the definitions of resource component of a subtree at a particular layer (i.e., the rectangular resource block) and resource interface of a subtree representing a collection of resource components.

**Definition 1** (Resource Component)**.** *The resource component of subtree $\mathbf{G}_{V_i}$ at layer $l$, denoted as $C_{i,l} = [n^s_{i,l}, n^c_{i,l}]$, specifies the set of consecutive cells required by all the links at layer $l$ in $\mathbf{G}_{V_i}$. $n^s_{i,l}$ and $n^c_{i,l}$ represent the number of cells in the time and channel dimensions, respectively.*

**Definition 2** (Resource Interface)**.** *The resource interface of subtree $\mathbf{G}_{V_i}$, denoted as $I_i = \{C_{i,l} | l = l(V_i), ...l(\mathbf{G}_{V_i})\}$, is the collection of resource components of $\mathbf{G}_{V_i}$ at all the layers.*

With the above definitions, the static partition allocation phase of HARP consists of two major operations. (1) *Resource interface generation*: Here starting from the non-leaf nodes that are farthest from the gateway, each non-leaf node $V_i$ generates the resource interface of subtree $\mathbf{G}_{V_i}$. Then resource interface $I_i$ is sent from $V_i$ to its parent node. This process

repeats until root node $V_g$ receives the resource interfaces of all its child nodes after which $V_g$ generates its own resource interface $I_g$. (2) *Partition allocation*: Here $V_g$ determines the partitions (i.e., the placement of each resource component in $I_g$ within the slotframe) to satisfy each resource component in $I_g$ and propagates the partition information to all its child nodes. Upon receiving the partition allocation from the parent node, each node $V_i$ allocates partitions for its subtrees at different layers within the partitions received from $V_i$'s parent.

### B. Generating Resource Interface

Resource interface generation is accomplished in the bottom-up fashion. According to Definition 1&2, in order to generate resource interface $I_i$, each non-leaf node $V_i$ needs to derive the resource components of subtree $\mathbf{G}_{V_i}$ at layers $l = l(V_i), ...l(\mathbf{G}_{V_i})$, i.e. computing $C_{i,l}, l \in \{l(V_i), l(V_i) + 1, ..., l(\mathbf{G}_{V_i})\}$. There are two cases for the computation of $C_{i,l}$.

**Case 1: Computation of $C_{i,l(V_i)}$.** $C_{i,l(V_i)}$ specifies the cells required by all the links $e_{i_1,i}, e_{i_2,i}, ..., e_{i_k,i}$ connecting $V_i$ and its $k$ child nodes $V_{i_1}, V_{i_2}, ..., V_{i_k}$ to satisfy their requirements, i.e., $r(e_{i_1,i}), ..., r(e_{i_k,i})$. Since links sharing a common node cannot be assigned to cells within a same time slot according to the constraint posted by the half-duplex mode of the radio, we can directly compute $n^s_{i,l(V_i)}$ as the accumulation of the number of cells required by all the links (i.e., $\sum_{m=1}^{k} r(e_{i_m,i})$) and let $n^c_{i,l(V_i)} = 1$. That is, $C_{i,l(V_i)} = [\sum_{m=1}^{k} r(e_{i_m,i}), 1]$.

**Case 2: Computation of $C_{i,l}$ where $l(V_i) + 1 \leq l \leq l(\mathbf{G}_{V_i})$.** Since $V_i$ does not directly have the cell requirement information of links in its subtrees, $V_i$ needs to generate the resource component $C_{i,l}$ by composing the resource interfaces of $V_i$'s direct subtrees. Specifically, suppose $V_i$ receives resource components $C_{i_1}, C_{i_2}, ..., C_{i_k}$ from its child nodes $V_{i_1}, V_{i_2}, ..., V_{i_k}$. Each resource components $C_{i_m} (m \in \{1, ..., k\})$ specifies the resource components of $\mathbf{G}_{V_{i_m}}$ at layers $l(V_i)+1, l(V_i)+2, ...l(\mathbf{G}_{V_{i_k}})$. Then $V_i$ needs to compose $k$ resource components $C_{i_1,l}, C_{i_2,l}, ..., C_{i_k,l}$ into one $C_{i,l}$ for each layer $l \in \{l(V_i) + 1, ..., l(\mathbf{G}_{V_i})\}$. Below, we describe in detail how $V_i$ performs the resource component composition.

Given that the total number of cells in a slotframe is limited, it is desirable to generate the resource component as compact as possible, i.e., minimizing the number of slots $n^s_{i,l}$ and the number of channels $n^c_{i,l}$ of $C_{i,l}$. For the following two reasons, we give a higher priority to minimizing the number of slots than the number of channels. First, reducing the number of time slots leads to shorter packet transmission latency. Second, time slots are more valuable than channels in the half-duplex mode because links sharing a common node cannot be assigned to cells within a same time slot. Therefore, we formulate the following resource component composition problem.

**Problem 1** (Resource Component Composition)**.** *Given $k$ resource components $C_{i_1,l}, C_{i_2,l}, ..., C_{i_k,l}$ at layer $l$, determine a composite resource component $C_{i,l} = [n^s_{i,l}, n^c_{i,l}]$ such that (i) $C_{i,l}$ contain all the $k$ components, i.e., $C_{i,l} \supseteq \cup_{m=1}^{k} C_{i_m,l}$, (ii) $n^s_{i,l}$ is minimized, and (iii) $n^c_{i,l}$ is minimized among all the components with the minimum $n^s_{i,l}$.*

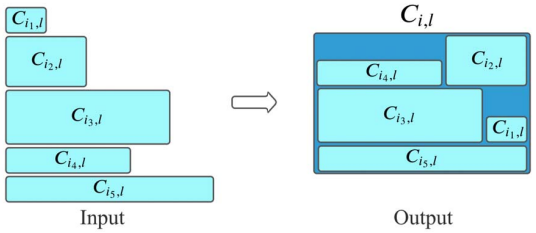



Fig. 4. An example resource component composition. Each light blue rectangle represents a resource component of subtree $\mathbf{G}_{V_{i_j}}$ at layer $l$.

Problem 1 can be mapped to a two-dimensional (2D) Strip Packing Problem (SPP) [20] as shown in Fig 4. SPP is a 2D geometric minimization problem. Specifically, given a set of axis-aligned rectangles and a strip with fixed width and infinite height, the objective is to determine an overlapping-free packing of the rectangles into the strip with the minimum height. To map Problem 1 to SPP, each resource component $C_{i_m,l}$ is mapped to a rectangle in SPP and the composite component $C_{i,l}$ is mapped to the strip. However, Problem 1 and SPP have a key difference. That is, the former aims to minimize both the number of slots (rectangle width) and the number of channels (rectangle height), while SPP only minimizes the height of the strip. Thus, directly applying an algorithm for solving SPP can cause the optimization goals (either the number of slots or channels) not being achieved.

We tackle this issue by performing the mapping twice. Since the total number of channels $M$ in IWNs is fixed (e.g., typically 16 channels for IEEE 802.15.4 in the 2.4GHz Band), we first map $M$ channels and the number of slots to the width and the height of the strip, respectively. By employing an SPP solver, a composite component with minimum number of slots $n^{min}_s$ is achieved. This generated component may use more channels than necessary since the width of the strip is not minimized. Therefore, we perform the second mapping where $n^{min}_s$ time slots and the number of channels are mapped to the width and the height of the strip, respectively. After solving this problem, the minimized strip is returned as the composite component $C_{i,l}$. Note that the generated packing of the rectangles which forms the layout of all the resource components is stored, and will be used by the partition allocation operation discussed in the next subsection.

SPP is a well-known NP-hard problem, and a large number of exact and approximation solvers exist in the literature (e.g. [21]–[23]). Considering that for our problem such a solver needs to be implemented in resource-constrained IWNs devices (e.g., TI CC2650), we opt to deploy an efficient heuristic. Among the various constructive heuristic candidates, the best-fit skyline based heuristic [24] is one of the state-of-the-art algorithms and achieves good balance between solution quality and efficiency (with time complexity of $O(n \log n)$ where $n$ is the number of rectangles). Alg. 1 summarizes the algorithm for solving Problem 1.

**Algorithm 1** Resource Component Composition

**Input**: Components $C_{i_1,l}, C_{i_2,l}, ..., C_{i_k,l}$, and $M$ channels.
**Output**: Composite component $C_{i,l}$ and composition layout.

1: Set $M$ channels as the fixed width, and let the number of time slots be the height in SPP;
2: Run the best-fit skyline heuristic and generate a component $C^*_{i,l}$ with $n_s^{min}$ slots and $n^{c\_slot}$ channels;
3: Set $n_s^{min}$ as the fixed width, and let the number of channels be the height in SPP;
4: Run the best-fit skyline heuristic and generate a component $C_{i,l}$ with $n_s^{min}$ slots and $n^{c\_min}$;
5: **return** $C_{i,l}$ and the composition layout;

### C. Partition Allocation

Given the resource interfaces generated at each non-leaf node in the IWN as described above, partition allocation determines the locations of these interfaces in the slotframe. We use $P_{i,l} = [C_{i,l}, t_{i,l}, c_{i,l}]$ to denote the partition allocated to subtree $\mathbf{G}_{V_i}$ at layer $l$ to satisfy its resource component $C_{i,l}$, where $t_{i,l}$ and $c_{i,l}$ represent the starting time slot and the lowest channel index of component $C_{i,l}$ in the slotframe.

In HARP, partition allocation is done in the top-down fashion starting from the root node $V_g$. Upon generating the resource components $C_{i,l}, C_{i,l(V_i)+1}, ..., C_{i,l(\mathbf{G}_{V_i})}$ of subtrees at all the corresponding layers, each node $V_i$ sends the collected resource interface $I_i$ to its parent node to request resource allocation. After the root node (i.e., gateway $V_g$) generates the resource interface $I_g$ including the components $C_{g,l}, l \in \{1, 2, ..., l(\mathbf{G}_{V_g})\}$, $V_g$ performs the partition allocation to satisfy each component $C_{g,l}$.

To ensure higher quality of service to the application tasks, a key objective in partition allocation is to place the interfaces such that they lead to low packet transmission latency. As stated in Sec. IV-A, the partition allocation at $V_g$ follows the compliant schedule property introduced in [19]. This property requires that cells are allocated to links following the sequence of a task's routing path. Specifically, the slotframe is divided into two super-partitions in the time dimension. The left and right super-partitions are allocated to the interfaces for uplinks and downlinks, respectively. Within the uplink (downlink) super-partition, any interface with a larger (smaller) layer value is placed before interfaces with smaller (larger) layer values. (see Fig. 2(c)).

According to the component composition layout of each $C_{g,l}$ generated by Alg. 1, $V_g$ can readily determine the partitions for all its direct subtrees at each layer $l$ (i.e., $P_{g_1,l}, P_{g_2,l}, ...$) within partition $P_{g,l}$. (See the upper sub-figure in Fig. 2(c) for an example partition allocation at $V_g$.) Then, $V_g$ propagates the partitions information for all its direct subtrees to $V_g$'s child nodes. When each node $V_i$ receives the partitions from its parent, $V_i$ repeats the partition allocation process and propagates the partitions to all its children. See the middle and bottom sub-figures in Fig. 2(c) for example partition allocation at subtree root nodes.

From Fig. 2(c), we can observe that after performing partition allocation, HARP achieves resource isolation between links that have different parent nodes. For example, links $e_{4,1}$ and $e_{7,3}$ are assigned to cells in partition $P_{1,2}$ and $P_{1,3}$, respectively. Links $e_{7,3}$ and $e_{9,5}$ are assigned to cells in partition $P_{3,3}$ and $P_{5,3}$, respectively. Generally, for any two links at layer $l_1$ and $l_2$, the assigned cells are within partition $P_{g,l_1}$ and $P_{g,l_2}$, so they are isolated from one another. Furthermore, for any two links in different subtrees at a same layer, the assigned cells are located in the partitions allocated to different subtrees.

### D. Distributed Schedule Generation

The static partition allocation phase discussed above determines the actual cells usable by each node. In the distributed scheduling phase, each node needs to handle traffic initiated by all its connected links in a distributed manner and the cells at each node needs to be further assigned to the links. To accomplish this, after receiving partition $P_{i,l(V_i)}$ from the parent, each node $V_i$ performs schedule generation to determine the cell assignment $\mathcal{C}(i,j)$ for each link $e_{i,j}$ connecting $V_i$ and its children at layer $l(V_i)$ within the cells in $P_{i,l(V_i)} = [[n^s_{i,l(V_i)}, 1], t_{i,l(V_i)}, c_{i,l(V_i)}]$. Based on the task set requirement, any scheduling policy can be deployed at $V_i$ to determine the particular schedule. In this work we apply Rate Monotonic (RM) [25], a well-known real-time scheduler, where the parent node selects cells from $P_{i,l(V_i)}$ according to the tasks' specifications and informs the cell assignment to corresponding child nodes. Since $n^s_{i,l(V_i)} \geq \sum_{m=1}^{k} r(e_{i_m,i})$, a feasible schedule can be constructed to satisfy the cell requirements of all the links $e_{i_m,i} (m \in \{1, 2, ..., k\})$.

## V. Dynamic Partition Adjustment

Network dynamics can vary due to various online events in harsh and noisy industrial environments, e.g., topology changes and traffic changes. From the network resource management's viewpoint, any type of changes may cause fluctuation in the cell requirements of certain links. If a node leaves the network or a task decreases the sampling rate, the number of cells required by some link may decrease. In this case, the parent node of the affected link readily releases the corresponding cells in the slotframe and the partitions of the subtree do not need to be adjusted. Therefore, in the following we focus on the network change scenario where cell requirements of certain links increase.

As discussed in Sec. IV-C, since HARP achieves resource isolation between links connecting to different parents, nodes are able to perform partition adjustment in a distributed manner. Thus we only need to consider how to perform partition adjustment at the parent node of a link requesting an increased number of cells. There are two cases.

**Case 1: Schedule update.** Parent node $V_q$ determines that the increased requirement can be satisfied in the current partition $P_{q,l(V_q)}$. That is, the total number of cells required by all the links connecting $V_q$ and its child nodes after the network change is not larger than $n^s_{q,l(V_q)}$. Fig. 5(a) is an

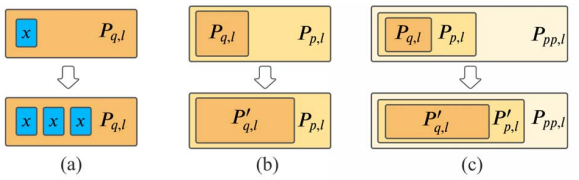



Fig. 5. Examples of partition adjustment under HARP: (a) local schedule adjustment; (b) partition adjustment finished within one layer; (c) partition adjustment crossed multiple layers.

example of this case. For this case, $V_q$ only needs to update the schedule within partition $P_{q,l(V_q)}$ and propagate the updates to the affected children.

**Case 2: Partition update.** $P_{q,l(V_q)}$ cannot satisfy the increased cell requirement. That is, the total number of cells required by all the links connecting $V_q$ and its child nodes after the network change is larger than $n^s_{q,l(V_q)}$. Fig. 5(b) and (c) are examples of this case.

Handling network changes in Case 2 is more complicated since $V_q$ must send its parent, say $V_p$, a request to increase partition $P_{q,l(V_q)}$ at layer $l(V_q)$ (i.e., an increased resource component $C_{q,l(V_q)}$). When $V_p$ receives the request, it checks if $C_{q,l(V_q)}$ can be satisfied within the allocated partition of subtree $\mathbf{G}_{V_p}$ with $V_p$ as root at layer $l(V_q)$. If so, $V_p$ accommodates the updated $C_{q,l(V_q)}$ by adjusting the partitions of subtrees with all $V_p$'s children (i.e., $V_q$'s siblings) as root nodes and sends the updated partition information to all the affected children. Otherwise, a partition update request is sent from $V_p$ to its parent, $V_{pp}$, following a similar process. Below, we describe i) how to check the feasibility of satisfying $C_{q,l(V_q)}$ and ii) how to perform partition adjustment to satisfy $C_{q,l(V_q)}$.

### A. Feasibility Test

Feasibility test is for a node to check whether it can satisfy within its own partition the increased resource requirement from one of its children. Formally, we have the following.

**Problem 2** (Feasibility Test)**.** *Given the updated component $C_{q,l(V_q)}$, the components $C_{p_m,l(V_q)}(m \in \{1,2,...,k,q\})$ where $V_{p_m}$ is a sibling of $V_q$, and the current partition $P_{p,l(V_q)}$, determine whether $P_{p,l(V_q)}$ can satisfy all the components.*

Problem 2 can be mapped to a classical Rectangle Packing Problem (RPP) [26] where the objective is to determine whether a given set of small rectangles can be packed inside a given large rectangle without any overlap. The mapping is straightforward where the components and the partition $P_{p,l(V_q)}$ are mapped to the small rectangles and the large rectangle, respectively. Then the best-fit skyline heuristic [24] can be applied to solve Problem 2 since both strip packing and rectangle packing are 2-dimensional geometric minimization problems sharing some similarities.

### B. Cost-aware Partition Adjustment

When node $V_p$ has to adjust its partition layout in the slotframe to accommodate a child's increased resource requirement, $V_p$ must send messages to all the children whose

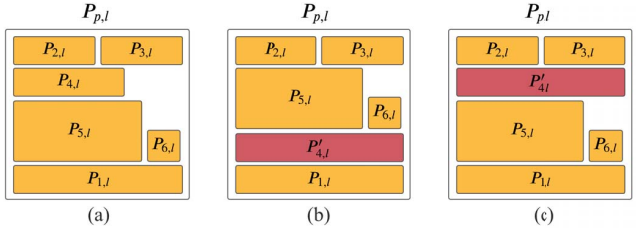



Fig. 6. Examples of partition adjustment. (a) Original partition layout. To accommodate an increased partition $P'_{4,l}$, (b) shows an adjusted partition layout with three updated partitions ($P'_{4,l}$, $P_{5,l}$ and $P_{6,l}$), and (c) shows an adjust partition layer with only one updated partition ($P'_{4,l}$).

partitions are updated. Furthermore, when a child node receives the updated partition, it needs to propagate the schedule updates messages to all the descendants. This is because that when a particular partition $P_{i,l}$ is changed in the slotframe, all the partitions inside $P_{i,l}$ are also changed. Propagating all the adjusted partitions can be quite costly. To reduce this communication overhead, i.e. the number of propagation messages, we formulate the following partition adjustment problem to minimize the number of adjusted partitions.

**Problem 3** (Partition Adjustment)**.** *Given $n$ resource components $C_{i,l}(i = 1..n)$ at layer $l$, the corresponding partitions $P_{i,l}$, the composite component $C_{p,l}$ and the corresponding partition $P_{p,l}$, assume that the $j$-th component $C_{j,l}(j \leq n)$ increases to $C'_{j,l}$ in either time slot or channel or both dimensions. Determine an updated partition $P'_{p,l}$ that satisfies $C'_{j,l}$ together with the unchanged component $C_{p,l}$ and minimizes the number of adjusted partitions $P_{i,l}(i = 1..n)$.*

Fig. 6 gives an example of the partition adjustment. Fig. 6(a) shows the original layout of partition $P_{p,l}$. Suppose that component $C_{4,l}$ increases in the time dimension and an updated partition $P'_{4,l}$ is needed to accommodate the increased component $C'_{4,l}$. Fig. 6(b) and Fig. 6(c) illustrates two feasible partitions which both satisfy all the components. However, the partition in Fig. 6(c) is apparently a better solution since all other partitions $P_{i,l}(i = 1..n, i \neq j)$ are not changed. While two additional partitions $P_{5,l}$ and $P_{6,l}$ are moved in Fig. 6(b) which incurs higher communication overhead.

We design an efficient heuristic to solve Problem 3. The heuristic is built on an intuitive observation that it tends to be easier for a consecutive area of idle cells to accommodate a set of partitions. Following this observation, the heuristic performs adjustment starting from the neighboring partitions of $P'_{j,l}$ then moving on to the more distant partitions.

Alg. 2 summarizes the partition adjustment algorithm for Problem 3. Specifically, we first replace $C_{j,l}$ with $C'_{j,l}$ and check whether the updated composite component can still fit in the given partition $P_{j,l}$ without adjusting any other partition $P_{i,l}(i = 1..n, i \neq j)$ (Line 1–4). If not, a randomly selected neighboring partition of $P'_{j,l}$ is removed from partition $P_{p,l}$, and we check whether $P'_{j,l}$ and the removed neighboring partition can be accommodated by using all idle rectangular areas in $P_{p,l}$ (Line 11, 12, 4). If yes, the new partition layout with only two adjusted partitions is returned. If not,

**Algorithm 2** Partition Adjustment Heuristic

**Input**: Components $C_{i,l}(i = 1, ..., n)$, partitions $P_{i,l}$, the composite $C_{p,l}$ and partition $P_{p,l}$, the increased component $C'_{j,l}(j \leq n)$.
**Output**: Updated partition $P'_{p,l}$.

1: $\mathcal{S} \leftarrow \{C'_{j,l}\}$;
2: Remove $P_{j,l}$ from $P_{p,l}$;
3: **while** True **do**
4: **if** All the components in $\mathcal{S}$ can be packed in $P_{p,l}$ **then**
5: $P'_{p,l} \leftarrow$ updated $P_{p,l}$;
6: **return** $P'_{p,l}$;
7: **else**
8: **if** $P_{p,l}$ is empty **then**
9: Break;
10: **end if**
11: $\mathcal{S} \leftarrow \mathcal{S} \bigcup \{C_{v,l}\}$ where $P_{v,l}$ is a partition closest to $P_{j,l}$ in $P_{p,l}$;
12: Remove $P_{v,l}$ from $P_{p,l}$;
13: **end if**
14: **end while**
15: Run the best-fit skyline heuristic for all the partitions;
16: **return** $P'_{p,l}$;

this process repeats until all the partitions $P_{i,l}(i = 1..n)$ are removed. Then, packing all the partitions into $P_{p,l}$ becomes the RPP discussed in Sec. V-A and the best-fit skyline heuristic can be applied to obtain the updated partition layout (Line 15).

## VI. Testbed Implementation and Validation

In this section, we present the implementation detail of the HARP framework on our 6TiSCH testbed and validate the functions of HARP in both static partition allocation phase and dynamic partition adjustment phase.

### A. Testbed Implementation and Experimental Setup

We implement HARP on a 5-hop 16-channel 6TiSCH network testbed with 50 devices. 6TiSCH is a representative multi-channel multi-hop IWN technology. It integrates the IEEE 802.15.4e 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 [27] 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. The 6TiSCH full stack is implemented on TI-RTOS [28] and deployed on both device nodes and the gateway. Fig. 7(a) shows the hardware used in our testbed and Fig. 7(b) gives a snapshot of the testbed deployment in labs and hallway. Fig. 7(c) depicts the logical topology of our network which is subject to change in our experiments. We set the slotframe length to 199 in the experiments and enable all the 16 IEEE 802.15.4e channels for communications. Fig. 7(d) shows the network schedule and partitioned slotframe.

When the network bootstraps, the slotframe is divided into two sub-frames: the Data sub-frame and the Management sub-frame. The Data sub-frame is the portion of the slotframe to be hierarchically partitioned and scheduled for real-time tasks deployed in the network. The Management sub-frame is the remaining portion of the slotframe used for scheduling network management traffic (e.g., Enhanced Beacon, RPL control messages, keep-alive packets). When a node joins the network, it will be first scheduled with two collision-free cells (one for uplink and the other for downlink) in the Management sub-frame. HARP messages are also transmitted in those cells.

TABLE I
CoAP handlers for HARP messages

| URI | Method | Param | Description |
|---|---|---|---|
| intf | POST | Resource interface | Receive child's interface |
| | PUT | Updated interface | Receive child's updated interface |
| part | POST | Partitions at all layers | Receive allocated partitions |
| | PUT | New partition at one layer | Receive updated partition |

Since HARP is an application layer protocol, we implement HARP messages and handlers on top of CoAP [29], which is a specialized HTTP-like web transfer protocol for resource constrained devices and used as the application layer of 6TiSCH. The defined CoAP handlers are summarized in Table I.

Fig. 8 depicts the flowchart of the HARP partition allocation and adjustment process. In the static partition allocation phase, the child nodes report their interfaces to their corresponding parent nodes by sending *POST-intf* messages. Once the parent node receives the interface information from all its child nodes, for each components in the interfaces, it runs the Resource Component Composition algorithm and reports the composite component to its own parent. This process repeats in a bottom-up fashion until reaches the gateway. Based on the received interface information from its child nodes, the gateway allocates partitions from the Data sub-frame and disseminates the partition information to its child nodes through the *POST-part* messages. Each child node stores the partition information of its own layer and further allocates partitions for its child nodes. Finally each individual non-leaf node is allocated with a dedicated partition, and any distributed scheduler can be employed to schedule the cells in this partition to satisfy the resource requirement without causing schedule collision. In the dynamic partition adjustment phase, a node that requires extra resources will send a partition adjustment request through a *PUT-intf* message to its parent node by specifying the new resource interface. The parent node then responds with the new partition by sending back a *PUT-part* message if it can satisfy the new request. Otherwise, it will forward the request to its parent in an iterative fashion until the request is satisfied.

To validate the functional correctness of HARP, we perform experiments in both static and dynamic network settings.

### B. Functional Validation on Static Partition Allocation

In the first set of experiments, we validate the functions of HARP to compose the resource interfaces in a hierarchical fashion and create dedicated partitions for individual subtrees to ensure deterministic communications.

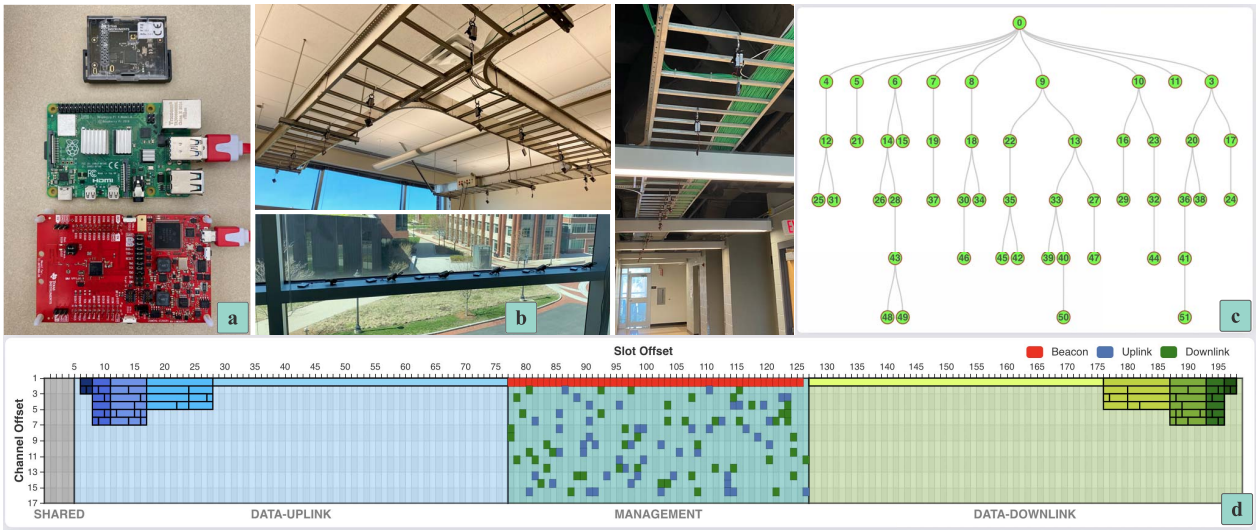



Fig. 7. (a) Hardware for the device nodes (CC2650) and gateway (RPi4B+CC2652); (b) an overview of the testbed deployed in the labs and hallways of an academic building; (c) the logical network topology with 5 layers; (d) an example slotframe with Data sub-frame and Management sub-frame.

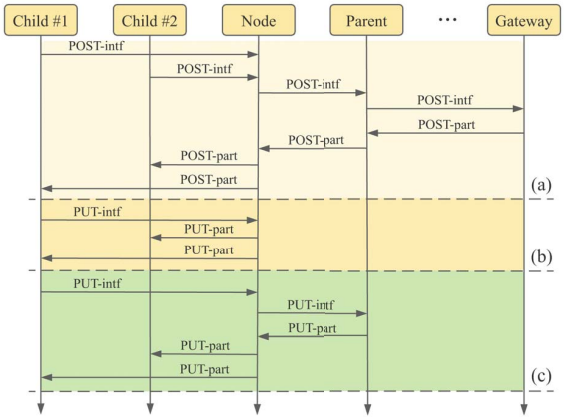



Fig. 8. HARP partition allocation and adjustment flowchart: (a) partition allocation; (b) one-hop partition adjustment; (c) multi-hop partition adjustment.

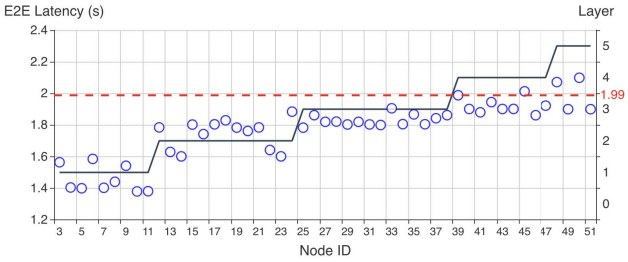



Fig. 9. End-to-end latency for all 50 nodes in the static network setup.

In the experiments, we configure the 50 device nodes into a 5-hop tree topology as shown in Fig. 7(c). We deploy an end-to-end (e2e) task with a period of 2 seconds on each individual node in the network. Each task transmits packets periodically to the gateway which echoes the packets back to the source device through the same routing path. Since the packet generation rates for all the device nodes are the same, the data rates of both uplink and downlink of individual nodes equal to the size of their subtrees, due to the fact that the parent nodes need to forward packets on behalf of their child nodes.

Under this experimental setting, the created partitions for individual subtrees are depicted in Fig. 7(d). The results are identical with those generated through simulation. This validates the correctness of HARP to compose resource interfaces and create partitions. We let the experiment run for 30 minutes, and collect the timing information of end-to-end packets from all the device nodes. The average end-to-end latency of each node are shown in Fig. 9 and the nodes are sorted according to the ascending order of their layers. From the results, we can observe that resources are dedicated for individual links based on their requirements and the measured end-to-end latency are almost bounded in one slotframe (1.99 seconds) with minimum queuing delay. However, it is worth noting that in the experiments we experienced some packet loss due to the environmental interference. This affects the latency for those nodes that are multiple hops away from the gateway.

### C. Functional Validation on Dynamic Partition Adjustment

In the second set of experiments, we validate the functions of HARP to perform partition adjustment in the presence of traffic changes. For this aim, during the network operation, we increase the data rates (in terms of packets per slotframe) of a selected set of nodes at different layers and monitor how their end-to-end Latency vary and what the associated communication overhead are. Fig. 10 summarizes the results.

Taking Node 15 as an exmaple, when the network starts, its data rate is first set at 1 packet/slotframe. This is the same for all the nodes in the network. The static partition allocation phase guarantees that Node 15's bandwidth requirement can be satisfied. There is no queuing delay and the transmissions can be finished within 1 slotframe (1.99 seconds). In the experiment, we increase the data rate of Node 15 to 1.5 packet/slotframe at time *02:25:00*. This increases the queuing

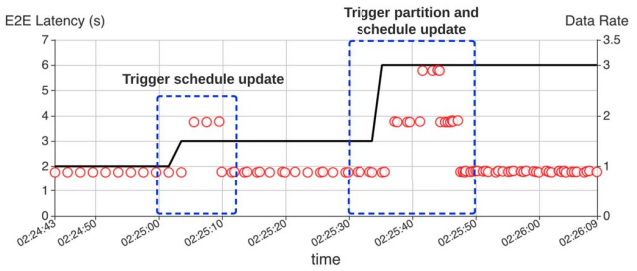



Fig. 10. End-to-end latency of Node 15 in the dynamic network setup, with the data rate increased from 1 packet/slotframe to 3 packets/slotframe.

TABLE II
COMPARISON OF THE PARTITION ADJUSTMENT OVERHEAD FOR A SELECTED SET OF NODES WITH DIFFERENT LAYERS

| Event | Nodes | Layers | Msg. | Time(s) | SF |
|---|---|---|---|---|---|
| $C_{5,2}$: $[1,1] \rightarrow [3,1]$ | 2 | 1 | 2 | 1.59 | 1 |
| $C_{22,3}$: $[3,1] \rightarrow [5,1]$ | 2 | 1 | 2 | 1.86 | 1 |
| $C_{3,2}$: $[7,1] \rightarrow [10,1]$ | 4 | 1 | 4 | 4.06 | 3 |
| $C_{10,3}$:$[2,2] \rightarrow [3,3]$ | 6 | 2 | 6 | 6.13 | 4 |
| $C_{40,5}$: $[1,1] \rightarrow [1,2]$ | 5 | 5 | 8 | 8.11 | 5 |
| $C_{30,4}$:$[1,1] \rightarrow [3,1]$ | 7 | 3 | 9 | 9.89 | 5 |

delay on Node 15 and its parent as well, and triggers them to allocate more cells from the allocated partition to increase the bandwidth. Since there are two idle cells available in the allocated partition, this traffic demand change can be resolved locally. At time *02:25:33*, the task rate of Node 15 is further increased to 3 packets/slotframe. However, this time there is no idle cells left in the allocated partition. This triggers the parent node to send a partition adjustment request to its parent to ask for more resources. After extra resource is granted, it updates the schedule between itself and Node 15 to meet the latest traffic requirement. This process takes longer time to adapt to the second traffic demand change, and thus cause longer queuing delay and increase the end-to-end latency.

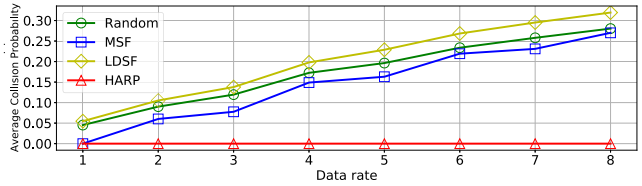
Table. II summarizes the interface update requests made by other nodes in the experiments, and their associated overheads in terms of involved nodes and layers, exchanged HARP messages and consumed time for partition adjustment. Due to the page limit, the detailed discussion is omitted.

## VII. SIMULATION-BASED PERFORMANCE EVALUATION

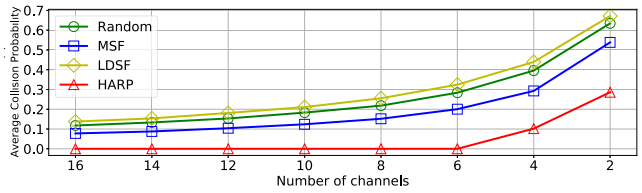
In this section, we present our simulation studies to evaluate the performance of HARP on avoiding schedule collisions and reducing schedule adjustment overhead when compared to the state-of-the-art distributed schedulers and centralized scheduler for 6TiSCH networks, respectively.

### A. Performance comparison on schedule collision avoidance

In the first set of experiments, we compare HARP with three distributed schedulers: random scheduler, MSF [10] and LDSF [30] deployed in 6TiSCH network, to evaluate their performance on providing collision-free communications. The random scheduler lets each node randomly select cell(s) in the slotframe for transmissions, while MSF allows the nodes to determine cell assignment based on a hash function of unique device IDs. LDSF divides the slotframes into small blocks




(a) Fixed number of channel, varying data rate




(b) Fixed data rate, varying number of channels

Fig. 11. Comparison of schedule collision avoidance among the random scheduler, MSF, LDSF and HARP.

and assign blocks to the links based on their layers to reduce latency, but the cell assignment within each block is random. We set the slotframe length to 199 time slots and enable all the 16 channels. We randomly generate 100 network topologies with 5 layers and 50 nodes.

We first compare the average collision probability among the random scheduler, MSF, LDSF and HARP by varying the data rates from 1 to 8 packet/slotframe, and the total number of cells required by all the nodes in a slotframe ranges from 150 to 700 in this setup. As shown in Fig. 11(a), the schedule collision probabilities of the random scheduler, MSF and LDSF increase linearly along with the increase of the data rates. This is due to the fact that more schedule collisions will occur when more cells are being assigned in the slotframe. On the other hand, HARP can always avoid transmission collisions in regardless of the data rate under these settings.

In the second comparison, we gradually reduce the number of available channels in the network from 16 to 2 while fixing the data rate at 3 packet/slotframe for all the nodes. As shown in Fig. 11(b), along with the decrease of the number of channels, the collision probabilities of the random scheduler, MSF and LDSF increase significantly. By contrast, benefiting from the efficient resource management, HARP is able to avoid collision when the number of channels is larger than 4. After that, its collision probability slightly increases but HARP still dominates the other three methods.

### B. Performance comparison on schedule adjustment overhead

In the second set of experiments, we compare the partition/schedule adjustment overheads between HARP and APaS [19] which is a centralized scheduler for 6TiSCH networks. In the experiments, we generate a series of network topologies with 81 nodes and 10 layers. After the static scheduling phase in APaS (static partition allocation phase in HARP), we increase the data rate of each node to trigger dynamic schedule update and partition adjustment phases in APaS and HARP, respectively, and measure the total number of packets incurred to complete the schedule/partition adjustment. Fig. 12 summarizes the results for the nodes at each layer.

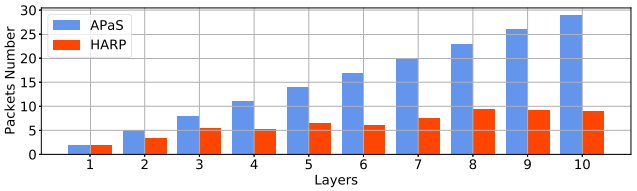



Fig. 12. Comparison of dynamic schedule/partition adjustment overhead between APaS and HARP

From the results, we can observe that the incurred adjustment overhead in APaS increases proportionally along with the increase of the layers. On the other hand, the adjustment overhead in HARP is relatively more stable. This is because in APaS, a node requesting for more resources needs to send the request to the root through multiple hops; the root then schedules new cells for this node and its parent node as well by sending back two schedule update messages through multiple hops as well. Thus for nodes at layer $l$, the total number of packets incurred in the dynamic schedule adjustment process is $3l-1$. On the other hand, the dynamic partition adjustment request in HARP is first sent to the parent of the requesting node. Only if the parent does not have sufficient resource to accommodate the request, the adjustment request is propagated upwards. Meanwhile, with the help of the partition adjustment heuristic (Alg. 2), the number of other affected branches is minimized. For these reasons, only a small portion of nodes in the network is involved in the partition adjustment phase, making HARP less sensitive to the size of the network compared to APaS.

## VIII. Conclusion and Future Work

In this work, we propose HARP, a hierarchical resource partitioning framework for dynamic resource management in IWNs. HARP achieves resource isolation between links by allocating dedicated cells for each link to support collision-free distributed scheduling. An efficient partition adjustment method is also proposed to adapt to network changes during runtime without introducing large communication overhead and time delay. We implement HARP on a 5-hop 50-node 6TiSCH network and validate its effectiveness through both simulation and testbed experiments. As future work, we will extend HARP to support non-tree based network topologies and real-time tasks with diverse end-to-end deadlines. We will also extend HARP to support dynamic resource management among co-existing heterogeneous IWNs.

## References


[1] 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," *Wirel. Netw.*, 2013.

[2] 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 *ECRTS*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[3] 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," *Proc. IEEE*, vol. 104, no. 5, pp. 1013–1024, 2015.

[4] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop IEEE 802.15.4e networks," in *PIMRC*. IEEE, 2012, pp. 327–332.

[5] 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*. IEEE, 2017, pp. 261–272.

[6] T. Zhang, T. Gong, S. Han, Q. Deng, and X. S. Hu, "Distributed dynamic packet scheduling framework for handling disturbances in real-time wireless networks," *IEEE Trans Mob Comput*, vol. 18, no. 11, pp. 2502–2517, 2018.

[7] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things," in *14th IEEE Int. Symp. World Wirel. Mob. Multimed. Netw. WoWMoM*. IEEE, 2013, pp. 1–6.

[8] 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 *RTAS*. IEEE, 2018, pp. 1–12.

[9] E. Municio, K. Spaey, and S. Latré, "A distributed density optimized scheduling function for IEEE 802.15. 4e TSCH networks," *ETT*, vol. 29, no. 7, p. e3420, 2018.

[10] "6TiSCH Minimal Scheduling Function (MSF) RFC9033," https://datatracker.ietf.org/doc/rfc9033/.

[11] A. J. Fahs, R. Bertolini, O. Alphand, F. Rousseau, K. Altisen, and S. Devismes, "Collision prevention in distributed 6TiSCH networks," in *WiMob*. IEEE, 2017, pp. 1–6.

[12] S. Kim, H.-S. Kim, and C. Kim, "ALICE: Autonomous link-based cell scheduling for TSCH," in *IPSN*, 2019, pp. 121–132.

[13] V. P. Modekurthy, A. Saifullah, and S. Madria, "DistributedHART: A distributed real-time scheduling system for wirelesshart networks," in *RTAS*. IEEE, 2019, pp. 216–227.

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

[15] S. C. Ergen, "ZigBee/IEEE 802.15. 4 Summary," *UC Berkeley, September*, vol. 10, no. 17, p. 11, 2004.

[16] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in *RTAS*. IEEE, 2008, pp. 377–386.

[17] I. Standard, "Wireless systems for industrial automation: process control and related applications," *ISA-100.11 a-2009*, p. 30, 2009.

[18] P. Djukic and S. Valaee, "Delay aware link scheduling for multi-hop tdma wireless networks," *IEEE ACM Trans Netw*, vol. 17, no. 3, pp. 870–883, 2008.

[19] J. Wang, T. Zhang, D. Shen, X. S. Hu, and S. Han, "APaS: An Adaptive Partition-based Scheduling Framework for 6TiSCH Networks," in *RTAS*. IEEE, 2021, pp. 320–332.

[20] B. S. Baker, E. G. Coffman, Jr, and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM J. Comput.*, vol. 9, no. 4, pp. 846–855, 1980.

[21] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi, "Exact algorithms for the two-dimensional strip packing problem with and without rotations," *Eur. J. Oper. Res.*, vol. 198, no. 1, pp. 73–83, 2009.

[22] K. Jansen and M. Rau, "Closing the gap for pseudo-polynomial strip packing," *arXiv preprint arXiv:1712.04922*, 2017.

[23] J. F. Oliveira, A. Neuenfeldt, E. Silva, and M. A. Carravilla, "A survey on heuristics for the two-dimensional rectangular strip packing problem," *Pesquisa Operacional*, vol. 36, pp. 197–226, 2016.

[24] L. Wei, Q. Hu, S. C. Leung, and N. Zhang, "An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation," *Comput Oper Res*, vol. 80, pp. 113–127, 2017.

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

[26] Y.-L. Wu, W. Huang, S.-c. Lau, C. Wong, and G. H. Young, "An effective quasi-human based heuristic for solving the rectangle packing problem," *Eur. J. Oper. Res.*, vol. 141, no. 2, pp. 341–358, 2002.

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

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

[29] "The Constrained Application Protocol (CoAP) RFC7252," https://datatracker.ietf.org/doc/html/rfc7252.

[30] V. Kotsiou, G. Z. Papadopoulos, P. Chatzimisios, and F. Theoleyre, "LDSF: Low-Latency Distributed Scheduling Function for Industrial Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8688–8699, 2020.