Composite Resource Scheduling for Networked Control Systems

Peng Wu*[§], Chenchen Fu[†][§], Tianyu Wang*, Minming Li[‡], Yingchao Zhao[††], Chun Jason Xue[‡], Song Han*
*University of Connecticut {peng.wu, tianyu.wang, song.han}@uconn.edu
[†]Southeast University chenchen_fu@seu.edu.cn
[‡]City University of Hong Kong {minming.li, jasonxue}@cityu.edu.hk
[††]Caritas Institute of Higher Education zhaoyingchao@gmail.com

***Abstract*—Real-time end-to-end task scheduling in networked control systems (NCSs) requires the joint consideration of both network and computing resources to guarantee the desired quality of service (QoS). This paper introduces a new model for composite resource scheduling (CRS) in real-time networked control systems, which considers a strict execution order of sensing, computing, and actuating segments based on the control loop of the target NCS. We prove that the general CRS problem is NP-hard and study two special cases of the CRS problem. The first case restricts the computing and actuating segments to have unit-size execution time while the second case assumes that both sensing and actuating segments have unit-size execution time. We propose an optimal algorithm to solve the first case by checking the intervals with 100% network resource utilization and modify the deadlines of the tasks within those intervals to prune the search. For the second case, we propose another optimal algorithm based on a novel backtracking strategy to check the time intervals with the network resource utilization larger than 100% and modify the timing parameters of tasks based on these intervals. For the general case, we design a greedy strategy to modify the timing parameters of both network segments and computing segments within the time intervals that have network and computing resource utilization larger than 100%, respectively. The correctness and effectiveness of the proposed algorithms are verified through extensive experiments.**

I. Introduction

Networked control systems (NCSs) are fundamental to many mission- and safety-critical applications that must work under real-time constraints to ensure timely collection of sensor data and on-time delivery of control decisions. The Quality of Service (QoS) offered by a NCS is thus often measured by how well it satisfies the end-to-end deadlines of the real-time tasks executed in NCSs [1]–[5].

A typical real-time task in NCSs involves both computing component(s) for executing the control algorithms on the controller and communication component(s) for exchanging sensor data and control signals between sensors/actuators and the controller. Traditional approaches to scheduling those tasks consider CPU and network resource scheduling separately. They either make an oversimplified assumption that the execution time of the control algorithm is negligible or consider it as constant. For example, extensive work has been reported in recent years on sensing and control task scheduling in real-time industrial networks [6]–[14]. Most of those work, however only focused on modeling the constraints of transmission conflicts but paid less attention to the computation time of the control algorithms when enforcing the end-to-end deadlines. Those approaches thus either cannot handle complex NCS applications which require the implementation of time-consuming control algorithms such as model predictive control using online optimization, data-driven system identification, and online learning control [15]–[19], or will lead to unnecessarily low utilization of the computing and network resources due to the lack of efficient methods to schedule those resources in a joint fashion. Take the F1/10 autonomous car system as an example that aims to race in a rectangular track while avoiding any obstacles [20]. The communication component on the system for exchanging sensing/actuating information has a worst-case execution time of 0.3 milliseconds; while the PID controller for steering control and the vision controller for identifying corners have worst-case execution times of 0.4 and 50 milliseconds, respectively. These comparable execution times of both networking and computing tasks motivate us to introduce a new task model, namely the Composite Resource Scheduling (CRS) model, for jointly scheduling network and computing resources in NCSs. In CRS, each real-time composite task consists of three consecutive and dependent segments: a sensing segment to transmit sensor data to the controller, a computing segment to compute the control decisions, and an actuating segment to transmit the control signals to the actuators. The sensing/actuating segments together are called network segments. They share the same network resource while the computing segments compete for the computing resource on the controller.

Similar models in the literature include PRedictable Execution Model (PREM) [21]–[23] and Acquisition Execution Restitution (AER) model [24]. These two models, however are specifically designed to schedule resources in multi-core systems with shared memory where the communication phases including memory read and write are considered as non-preemptive, which is reasonable for the design of multi-core systems. By contrast, the network segments of the CRS model for NCSs are designed to be preemptive to meet the flexibility of the network scheduling. This provides better parallelism when scheduling sensing/actuating segments and computing segments from different tasks.

Our work attempts to tackle the composite resource scheduling problem in NCSs, which aims to optimize the usage of network and computing resources in NCSs under the end-to-end deadline constraints. Based on the proposed CRS model, we provide a comprehensive analysis on the complexity of the problem, and prove that the general CRS problem is

[§]The first two authors are the corresponding authors.

NP-hard in the strong sense. We thus in this paper study two special cases of the CRS problem and present a greedy heuristic for the general cases as well, focusing on the design of the corresponding algorithms to construct feasible composite schedules. We start with the first case where both the computing segment and actuating segment have unit-size execution time. In this case, an optimal scheduling algorithm is proposed. The algorithm first modifies the deadlines of the CRS task set based on the intervals with 100% network resource utilization and then applies the Earliest Deadline First (EDF) scheduling algorithm to schedule the sensing and actuating segments together according to the modified deadlines. For the second case where the execution time of both sensing and actuating segments are unit-size, and the execution time of the computing segment can be an arbitrary integer larger than one, we design another optimal scheduling algorithm which schedules the computing segments in the first stage and the sensing and actuating segments in the second stage. If the scheduling fails in the second stage, the algorithm rolls back to the initial stage based on a novel backtracking search strategy by adding new constraints to the original problem recursively, and eventually lead to a feasible composite schedule if it exists. For the general case, we propose a heuristic solution with a roll-back mechanism. EDF is first employed to schedule the segments. If EDF fails to find a feasible schedule, we locate the intervals with either network resource or computing resource utilization larger than 100% and modify the deadline of the segment included in the located interval with the earliest release time. We iteratively run EDF and modify the timing parameters until we find a feasible schedule. The effectiveness of the proposed algorithms has been validated through extensive experiments. Our results show that the proposed scheduling algorithms outperform the baseline algorithms in terms of schedulability.

The remainder of this paper is organized as follows: Section II presents the task model and the formulation of the general CRS problem and its two special cases. Section III and Section V develop the optimal scheduling algorithms for the first and second cases, respectively. Section V introduces a heuristic scheduling algorithm for the general case. Section VI presents the experimental results. Section VII gives a summary of the related works. Section VIII concludes the paper and discusses the future work.

II. Task Model and Problem Formulation

In this section, we introduce the CRS model and present the constraint programming formulation of the composite resource scheduling problem. Based on different settings on the execution time of the network/computing segments, we introduce three CRS models.

A. Task Model

Consider a NCS configured in a star topology. The sensors and actuators form the leaf nodes while the controller runs on the root (Gateway). To ensure efficient and safe operation of the control system, the transmissions of sensor data/control signals and the execution of the control algorithm should follow a strict order and meet the designated end-to-end timing requirement. In a typical real-time composite task in NCSs, one or multiple sensors send their measurements to the controller first. The controller then computes the control signals and delivers them to one or multiple actuators. In practice, while the CPU is waiting for the sensing data, it can be set to idle. Once the transmission of sensing data is finished, an interrupt notifies the CPU to obtain the data stored in a specified shared memory. This decouples the computation from sensing and actuating so that the transmissions of sensing data and control signals share the network resource while the executions of the control algorithms only compete for the CPU resource. In our CRS model, we consider multiple-input multiple-output (MIMO) control systems, which have multiple sensors as the input and multiple actuators as the output. As shown in Fig. 1, each real-time composite task $\tau_i = (T_i, D_i, C_{i,s}, C_{i,c}, C_{i,a})$ is associated with a period T_i and a relative deadline D_i. It consists of three consecutive and dependent segments:

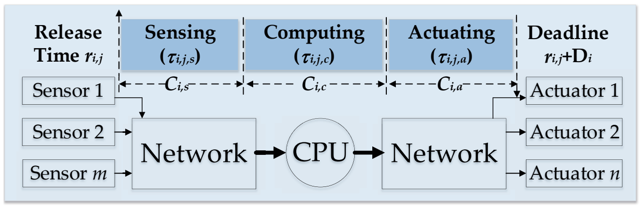


Fig. 1: An overview of the real-time composite task model

- Sensing segment: it utilizes the network resource and has the transmission time of $C_{i,s}$;
- Computing segment: it utilizes the CPU resource and has the execution time of $C_{i,c}$;
- Actuating segment: it utilizes the network resource and has the transmission time of $C_{i,a}$.

In this paper, we consider a time-slotted system for network. A network segment consists of multiple fragments where each fragment takes exactly one time slot in the super-frame to transmit. We define the execution times $C_{i,s}$ and $C_{i,a}$ of the sensing and actuating segments to be the sum of the network transmission times of all sensors and actuators connected to a control task, and that their activation frequency is equal to the one of the task. We assume that (1) the unit sizes of the network and computing resources are the same; (2) the network is single-channel and the CPU is a preemptive uniprocessor; (3) the task system is a synchronous system; and (4) the release time, deadline and execution time are integers.

B. Problem Formulation

Based on the above task model, we now present the formal definition of the composite resource scheduling problem and give its constraint programming formulation.

Composite Resource Scheduling (CRS) Problem: Consider a set of real-time composite tasks $\{\tau_1, \tau_2, ..., \tau_n\}$ with $\tau_i = (T_i, D_i, C_{i,s}, C_{i,c}, C_{i,a})$ and the hyper-period $\mathcal{H}$ obtained by computing the least common multiple of the periods. The objective of the CRS problem is to find a feasible composite schedule with a length of $\mathcal{H}$ if it exists so that the deadlines of all the real-time composite tasks are met.

We first formulate the CRS problem as a constraint programming problem. It aims to construct the feasible schedules $\mathcal{S}_{\text{net}} = \{f_{i,j,s,k}, f_{i,j,a,k}\}$ for the network segments and $\mathcal{S}_{\text{com}} = \{f_{i,j,c,k}\}$ for the computing segments, where $f_{i,j,s,k}$, $f_{i,j,c,k}$, and $f_{i,j,a,k}$ are the finish times of the k^{th} unit of sensing/computing/actuating segments of the j^{th} instance of the task τ_i, respectively. The j^{th} instance of task τ_i, denoted as $\tau_{i,j}$, has a release time $r_{i,j}$ and a deadline $r_{i,j} + D_i$. The construction of the feasible schedules is subject to the following constraints:

Release time and deadline constraints:

$$\begin{aligned} f_{i,j,s,1} &\geq r_{i,j} + 1 \\ f_{i,j,a,C_{i,a}} &\leq r_{i,j} + D_i \end{aligned} \tag{1}$$

Segment order constraints:

$$\begin{aligned} f_{i,j,c,1} &\geq f_{i,j,s,C_{i,s}} + 1 \\ f_{i,j,a,1} &\geq f_{i,j,c,C_{i,c}} + 1 \\ f_{i,j,s,k+1} &\geq f_{i,j,s,k} + 1, \quad \forall k \in \mathbb{Z} \cap [1, C_{i,s} - 1] \\ f_{i,j,c,k+1} &\geq f_{i,j,c,k} + 1, \quad \forall k \in \mathbb{Z} \cap [1, C_{i,c} - 1] \\ f_{i,j,a,k+1} &\geq f_{i,j,a,k} + 1, \quad \forall k \in \mathbb{Z} \cap [1, C_{i,a} - 1] \end{aligned} \tag{2}$$

Network resource constraints:

$$\begin{aligned} &\forall i \neq z, \quad \forall x, y \in \{a, s\} \\ &\forall p \in \mathbb{Z} \cap [1, C_{i,x}], \quad \forall q \in \mathbb{Z} \cap [1, C_{z,y}] \\ &f_{i,j,x,p} \leq f_{z,j,y,q} - 1 \quad \text{OR} \quad f_{i,j,x,p} \geq f_{z,j,y,q} + 1 \end{aligned} \tag{3}$$

Computing resource constraints:

$$\begin{aligned} &\forall i \neq z, \quad \forall p \in \mathbb{Z} \cap [1, C_{i,c}], \quad \forall q \in \mathbb{Z} \cap [1, C_{z,c}] \\ &f_{i,j,c,p} \leq f_{z,j,c,q} - 1 \quad \text{OR} \quad f_{i,j,c,p} \geq f_{z,j,c,q} + 1 \end{aligned} \tag{4}$$

Constraint (1) requires that the first unit of the sensing segment and the last unit of the actuating segment of any task τ_i are constrained by τ_i's release time and deadline, respectively. Constraint (2) defines the constraints for the sequential order for each unit of sensing, computing, and actuating segments. One can easily derive the earliest possible release time and the latest possible deadline for each segment based on Constraints (1) and (2). Constraints (3) and (4) define the constraints for different composite tasks to compete for the network and computing resources, where the OR operator in the constraints can be modeled by using the big M method [25]. Note that the above formulation employing the time-indexed model [26] is computationally efficient since the problem studied in the paper is a preemptive flow shop scheduling problem.

TABLE I: Problems and solutions under different CRS models.

Task Model	Complexity or Solution
h-1-1	Exponential-time solvable (Alg. 1)
1-m-1	Exponential-time solvable (Alg. 2)
h_1-h_2-h_3	NP-hard and solved by heuristics (Alg. 4)

To study the CRS problem comprehensively, we consider three cases of the CRS model based on the size of the execution time of each segment. These cases are summarized in Table I. The first case is the h-1-1 model where $C_{i,c} = C_{i,a} = 1$ and $C_{i,s} = h$ ($h > 0$) for each segment of a task. The second case uses the 1-m-1 model with $C_{i,s} = C_{i,a} = 1$ and $C_{i,c} = m$ ($m > 1$). The third case is the general model h_1-h_2-h_3 with $C_{i,s} = h_1$, $C_{i,c} = h_2$, and $C_{i,a} = h_3$ ($h_1, h_2, h_3 > 0$). These three cases represent three types of application scenarios in the real life. While $h-1-1$ refers to the task which has a longer sensing segment, $1-m-1$ represents the task which has a longer computing segment, and h_1-h_2-h_3 represents the general applications. The following theorem shows that the CRS problem under the general model is NP-hard in the strong sense.

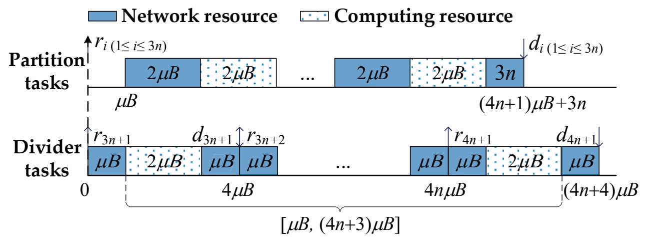


Fig. 2: An instance of the composite schedule

Theorem 1. *The CRS problem under the general model is NP-hard in the strong sense.*

Proof. We reduce 3-Partition to the CRS problem. An instance of 3-Partition consists of a list $A = (x_1, x_2, ..., x_{3n})$ of positive integers such that $\sum x_i = nB$, $\frac{B}{4} < x_i < \frac{B}{2}$ for each $1 \leq i \leq 3n$, there exists a partition of A into A_1, A_2,...,A_n such that $\sum_{x_i \in A_k} x_i = B$ for each $1 \leq k \leq n$ [27].

Given an instance $A = (x_1, x_2, ..., x_{3n})$ of 3-Partition, we construct an instance of the CRS problem as follows. As shown in Fig. 2, there will be $3n + 1$ tasks. The first $3n$ tasks are *partition* tasks. For each $1 \leq i \leq 3n$, the task τ_i satisfies that $C_{i,s} = 2\mu x_i$, $C_{i,c} = 2\mu x_i$, $C_{i,a} = 1$, $D_i = (4n + 1)\mu B + 3n$, and $T_i = 4\mu B$, where B is the sum of each partition and $\mu = \lceil \frac{3n}{B} \rceil$. The last task is *divider* task which satisfies that $T_i = D_i = 4\mu B$, $C_{i,s} = \mu B$, $C_{i,c} = 2\mu B$, and $C_{i,a} = \mu B$. The $n + 1$ divider tasks divide the timeline of $[\mu B, (4n + 3)\mu B]$ into $n + 1$ intervals of full computing resource utilization interleaved with n intervals of full network resource utilization, where the partition tasks are scheduled. The length of each of these intervals is exactly $2\mu B$. The sensing segments of the partition tasks from the same partition must be scheduled in a length $2\mu B$ of the interval $[(2j - 1)\mu B, (2j + 1)\mu B]$ so that their corresponding computing segments can anticipate releasing exactly at or before $(2j+1)\mu B$ and fully utilizing the interval $[(2j + 1)\mu B, (2j + 3)\mu B]$, where $1 \leq j \leq n$. Thus, it is easy to see that there is a feasible schedule of the CRS problem if and only if there exists a 3-Partition. It is clear that the above reduction is polynomial. We thus have proved that the CRS problem is NP-hard in the strong sense. □

We first exploit the earliest possible release time and the latest possible deadline to define the *effective timing parameters* of the network and computing segments.

Definition 1. Effective Release Time/Deadline: *Given an instance of real-time composite task $\tau_{i,j}$ with the release time $r_{i,j}$ and deadline $d_{i,j}$, the effective release time/deadline of its sensing segment $\tau_{i,j,s}$ are defined as $\overline{r}_{i,j,s} = r_{i,j}$ and $\overline{d}_{i,j,s} = d_{i,j} - C_{i,a} - C_{i,c}$. The effective release time/deadline of its computing segment $\tau_{i,j,c}$ are defined as $\overline{r}_{i,j,c} = r_{i,j} + C_{i,s}$ and $\overline{d}_{i,j,c} = d_{i,j} - C_{i,a}$. The effective release time/deadline of its actuating segment $\tau_{i,a}$ are defined as $\overline{r}_{i,j,a} = r_{i,j} + C_{i,s} + C_{i,c}$ and $\overline{d}_{i,j,a} = d_{i,j}$.*

We say a network segment is *effectively included* in a time interval if its effective release time and effective deadline are within that interval. For instance, a network segment $\tau_{i,j,\text{net}}$ is effectively included in $[t_0, t_1]$ if $\overline{r}_{i,j,\text{net}} \geq t_0$ and $\overline{d}_{i,j,\text{net}} \leq t_1$. Based on this condition, we define an *effective network demand* over a given interval to be the sum of the execution time of all network segments that are effectively included in that interval. We define the *effective network overload interval* and *effective network tight interval* as follows.

Definition 2. Effective Network Overload/Tight Interval (ENOI/ENTI): *Given a set of real-time composite tasks and a time interval $[t_0, t_1]$, $[t_0, t_1]$ is an effective network overload interval if the effective network demand over $[t_0, t_1]$ is larger than $t_1 - t_0$. $[t_0, t_1]$ is an effective network tight interval if the effective network demand over $[t_0, t_1]$ is equal to $t_1 - t_0$.*

Since the CRS problem under the general model is NP-hard, in the following we first focus on the design of exact algorithms for the first two simple models and then present an effective heuristic solution for the general model.

III. CRS Problem under H-1-1 Model

The h-1-1 model represents a wide range of NCSs which need to transport a large amount of sensing data while the required computing and actuation time are short. In the following, after giving several important observations, we present an exact optimal algorithm to solve it.

Based on the effective timing parameters, we employ EDF to schedule the real-time composite tasks in the following fashion. We utilize a network ready queue and a computing ready queue for scheduling the network segments and computing segments, respectively. Once a sensing segment $\tau_{i,j,s}$ is finished at $f_{i,j,s}$, its computing segment $\tau_{i,j,c}$ is released at $f_{i,j,s}$. Similarly, if a computing segment $\tau_{i,j,c}$ is finished at $f_{i,j,c}$, its actuating segment $\tau_{i,j,c}$ is released at $f_{i,j,c}$. The actuating segments will be scheduled together with all other network segments (including both sensing and actuating segments from other task instances) using EDF. The effective deadlines of the sensing and actuating segments are used to decide their priorities. Ties are broken by scheduling the sensing segments first. For the same type of segments, ties are broken by scheduling the segment with the least laxity.

Lemma 1. *Given a real-time composite task set $\mathcal{T}$ under the h-1-1 model, if EDF can find a feasible network schedule for the sensing segments based on their effective deadlines, it can also find a feasible computing schedule based on the effective deadlines of their computing segments.*

It is straightforward to prove the correctness of Lemma 1. As the finish times of the sensing segments are different, the release times of their corresponding computing segments are different as well. Given that each computing segment takes unit-size execution time, we can always schedule the computing segments immediately after the completion of its corresponding sensing segments. Lemma 1 indicates that scheduling the sensing segments does not interfere with scheduling the computing segments. However, it is hard to guarantee that EDF can also construct a feasible schedule for the actuating segments as they compete for the network resource with the sensing segments. Therefore, in the following we focus on adapting EDF to schedule the sensing and actuating segments.

The following lemma gives a necessary condition to determine the schedulability of the real-time composite task set under the h-1-1 model.

Lemma 2. *Given a real-time composite task set $\mathcal{T}$ under the h-1-1 model, if there exists an effective network overload interval (ENOI), the task set is unschedulable.*

Proof. Due to the page limit, please refer to the full technical report for the proof [28] □

An effective network tight interval (ENTI) indicates that this interval has a 100% network resource utilization in any feasible schedule. Thus, it is important to utilize the ENTIs to modify the deadline of the tasks whose deadlines are within the interval but their actuating segments are not effectively included in the interval. The following definition and lemma show that EDF can find the feasible schedule if there exists one after applying the ENTIs to modify the task deadlines.

Definition 3. Continuous Interval: *Given a real-time composite task set $\mathcal{T}$ under the h-1-1 model, suppose $\tau_{z,j,\text{net}}$ is the first network segment scheduled by EDF to miss its deadline $\overline{d}_{z,j,\text{net}}$ and b_0 is the start time of a network segment in the schedule, we define $[b_0, \overline{d}_{z,j,\text{net}}]$ to be a continuous interval if it is fully utilized by the network segments and there exists no network segment scheduled in the interval $[b_0 - 1, b_0]$ or the network segment scheduled in $[b_0 - 1, b_0]$ has a deadline later than $\overline{d}_{z,j,\text{net}}$.*

Lemma 3. *Given a schedulable real-time composite task set $\mathcal{T}$ under the h-1-1 model, if EDF fails to find a feasible network schedule, then there exists exactly one actuating segment scheduled in a continuous interval but not effectively included in that interval, and the continuous interval is an ENTI.*

Proof. Based on proof by contradiction, we assume that one of the following cases holds.

- **Case 1**: the network segments scheduled in the continuous interval are all effectively included in that interval.
- **Case 2**: there exists exactly one sensing segment scheduled in the continuous interval but not effectively included in it.
- **Case 3**: there exist at least two network segments scheduled in the continuous interval but not effectively included in it.

Suppose that EDF generates the network schedule $\mathcal{S}_{\text{net}}$. Consider that EDF fails to find a feasible network schedule. Let $\tau_{j,\text{net}}$ be the first network segment scheduled by EDF that misses its deadline $\overline{d}_{z,j,\text{net}}$. There exists the continuous interval $[b_0, \overline{d}_{z,j,\text{net}}]$ which is fully utilized by the network segments. There exists no network segment scheduled in the interval $[b_0 - 1, b_0]$ or the network segment scheduled in it has a deadline later than $\overline{d}_{z,j,\text{net}}$.

If case 1 holds, all the network segments scheduled in $[b_0, \overline{d}_{z,j,\text{net}}]$ are effectively included in the interval, with the network segment $\tau_{z,j,\text{net}}$ considered, the interval $[b_0, \overline{d}_{z,j,\text{net}}]$ is an ENOI, which is a contradiction to that $\mathcal{T}$ is schedulable according to Lemma 2.

If case 2 holds, since there exists exactly one sensing segment scheduled in the continuous interval but not effectively included in the interval, this sensing segment should be scheduled in $[b_0 - 1, b_0]$ according to EDF. This contradicts to the definition of continuous interval.

If case 3 holds, there exist at least two network segments whose corresponding effective release times are smaller than b_0 among the network segments scheduled in $[b_0, \overline{d}_{z,j,\text{net}}]$. If at least one of the two network segments is a sensing segment, the interval $[b_0 - 1, b_0]$ must be utilized by this sensing segment, which is a contradiction. Therefore, we consider the case that both of them are actuating segments. Let $\tau_{k,x,a}$ and $\tau_{g,y,a}$ be the two actuating segments satisfiying $\overline{r}_{k,x,a}, \overline{r}_{g,y,a} < b_0$. For their corresponding sensing segments $\tau_{k,x,s}$ and $\tau_{g,y,s}$, it must hold that $\overline{r}_{k,x,s}, \overline{r}_{g,y,s} < b_0 - 1$. Thus, if any of the two sensing segments $\tau_{k,x,s}$ and $\tau_{g,s}$ are scheduled in $[b_0, \overline{d}_{z,j,\text{net}}]$, it is a contradiction that the interval $[b_0 - 1, b_0]$ is not utilized or by a network segment with the deadline later than $\overline{d}_{z,j,\text{net}}$. Hence, both $\tau_{k,x,s}$ and $\tau_{g,y,s}$ must be scheduled before $b_0 - 1$. Since the finish times obtained by EDF must be different, at least one of $f_{k,x,s}$ and $f_{g,y,s}$ is smaller than $b_0 - 1$. Therefore, at least one of the actual release times of $\tau_{k,a}$ and $\tau_{g,y,a}$ is smaller than b_0. This allows one of $\tau_{k,x,a}$ and $\tau_{g,a}$ to be scheduled in $[b_0 - 1, b_0]$, but leads to a contradiction to our assumption. Therefore, there exists exactly one actuating segment whose effective release time is smaller than b_0. With the network segment $\tau_{z,j,\text{net}}$ considered, the interval $[b_0, \overline{d}_{z,j,\text{net}}]$ is an ENTI. This proves that there exists exactly one actuating segment scheduled in the ENTI $[b_0, \overline{d}_{z,j,\text{net}}]$ but not effectively included in it if EDF fails. ☐

Lemma 3 indicates that as long as the actuating segments which do not belong to a given ENTI are prevented from being scheduled inside that interval, EDF can construct a feasible network schedule if it exists.

Based on Lemma 3, Alg. 1 gives an overview of the CRS algorithm under the h-1-1 model. The algorithm first constructs the effective timing parameters for all the segments. Based on these effective release times and deadlines, the algorithm identifies every ENTI and modifies the effective timing parameters accordingly. To ensure that the search of the ENTIs is complete, we traverse the effective release times of all sensing segments in the descending order, and the effective deadlines of all actuating segments in the ascending order to construct all candidate intervals. If an ENTI $[t_0, t_1]$ is found, we check all the tasks to modify their deadlines based on the following rule: if the deadline is included in $[t_0, t_1]$ and the effective release time of its actuating segment is smaller than t_0, we set its deadline to t_0. Additionally, the algorithm identifies the ENOI and returns a None value to indicate that this task set is unscheduable. This procedure (line 3-16 in Alg. 1) repeats until all ENTIs are identified, which has a time complexity of $O(N^3)$, where $N = \sum_{i=1}^{n} \mathcal{H}/T_i$ is the total number of instances obtained from all the tasks. After that, we employ EDF to construct a feasible schedule. The overall time complexity of Alg. 1 is $O(N^3)$.

Algorithm 1: CRS Algorithm (h-1-1 Model)

```
Input  : A real-time composite task set $\mathcal{T} = \{\tau_i\}_{i=1}^n$
Output: A network schedule $\mathcal{S}_{\text{net}}$ and a computing schedule
        $\mathcal{S}_{\text{com}}$, if exist
1  Compute the hyper-period $\mathcal{H}$ and build the set $I$ of instances
     of $\mathcal{T}$
2  Construct the set $O$ of effective network overload and tight
     intervals
3  while $O \neq \emptyset$ do
4      if $\alpha \in O$ is an ENOI then
5          return None // Algorithm reports a failed
               case
6      end
7      if $\lambda \in O$ is an ENTI $[t_0, t_1]$ then
8          for $\tau_{i,j} \in I$ do
9              if $d_{i,j} \in \lambda$ and $\overline{r}_{i,j,a} < t_0$ then
10                 $d_{i,j} = t_0$
11             end
12         end
13         $O = O - \{\lambda\}$
14     end
15     Update the set $O$ based on the modified timing
         parameters
16 end
17 Use EDF to schedule the network and computing segments
     in parallel based on their effective deadlines to obtain the
     schedules $\mathcal{S}_{\text{net}}$, and $\mathcal{S}_{\text{com}}$
18 return $\mathcal{S}_{\text{net}}$, $\mathcal{S}_{\text{com}}$
```

The following theorem proves the correctness of the CRS algorithm under the h-1-1 model.

Theorem 2. *If there exists a feasible schedule for the real-time composite task set under the* h-1-1 *model, then Alg. 1 can find it.*

Proof. The proposed CRS algorithm under the h-1-1 model modifies the task deadlines so that there exist no actuating segments which would be scheduled in an ENTI if they are not effectively included in the interval. According to Lemma 3, if the single actuating segment which is not included in the ENTI is removed, the EDF is able to find a feasible network schedule $\mathcal{S}_{\text{net}}$ for the network segments. Besides, according to Lemma 1, the computing schedule $\mathcal{S}_{\text{com}}$ must be feasible as well. ☐

IV. CRS Problem under 1-M-1 Model

We now extend the study to the 1-m-1 model, where every real-time composite task has unit-size execution time for its sensing/actuating segments and arbitrary execution time larger than 1 for its computing segment. This model represents a wide range of NCSs which employ online optimization methods so that their computing time is longer than the transmission time for sensing and actuating. Different from the h-1-1 model, infeasible schedules under the 1-m-1 model can be generated by scheduling the computing segments improperly. Thus, it is important to take into account the scheduling of computing segments in the algorithm design.

In this section, we present an exponential-time optimal algorithm to solve the CRS problem under the 1-m-1 model based on a novel backtracking strategy. The effectiveness of the proposed algorithm has been validated through extensive experimental results (see Section VI).

A. Algorithm Overview

Alg. 2 gives an overview of the CRS algorithm under the 1-m-1 model, which utilizes an iterative two-stage decomposition method. We decompose the CRS problem into two sub-problems including the computing scheduling sub-problem and the network scheduling sub-problem. The variables of the original CRS problem are also divided into a subset of computing variables and a subset of network variables. The first-stage sub-problem is solved over the computing variables. The values of the network variables are determined in the second-stage sub-problems based on the given first-stage solution. If the subsequent sub-problem determines that the previous stage' decisions lead to infeasible schedules, then new constraint(s) will be added to the original CRS problem, which is re-solved until no new constraints can be added. Failure will be reported if no feasible composite schedule can be constructed.

Based on the algorithm framework above, it is important to guarantee that the new constraints added in each iteration will not jeopardize the schedulability of the original CRS problem. To tackle this challenge, we design a constraint generator to modify the timing parameters of the tasks in the interval(s) with the network resource utilization larger than 100% based on the iterative two-stage decomposition method.

B. Design details of the CRS algorithm under 1-m-1 model

1) Decomposition Method: We now reformulate the original CRS problem as a two-stage scheduling problem.

Computing Scheduling Sub-Problem: Consider a real-time composite task set $\mathcal{T}$ and the hyper-period $\mathcal{H}$. The objective of the computing scheduling sub-problem is to find a feasible computing schedule $\mathcal{S}_{com}$ with a length of $\mathcal{H}$ if it exists so that the effective deadlines of all computing segments are met.

Network Scheduling Sub-Problem: Given a real-time composite task set $\mathcal{T}$, the hyper-period $\mathcal{H}$ and the computing schedule $\mathcal{S}_{com}$, the objective of the computing scheduling sub-problem is to find a feasible network schedule $\mathcal{S}_{net}$ with a length of $\mathcal{H}$ if it exists so that the network segments can meet the deadlines obtained based on the start times and finish times of the computing segments in $\mathcal{S}_{com}$.

Since each sub-problem above can be taken as a uniprocessor scheduling problem, it is intuitive to design a two-stage EDF to solve the CRS problem by solving the computing scheduling sub-problem in the first stage and the network scheduling sub-problem in the second stage. However, the two-stage EDF may not find a feasible schedule for the second-stage problem as it always employs a fixed first-stage solution. Therefore, we utilize a constraint generator to add new constraints to the original CRS problem whenever two-stage EDF finds an infeasible schedule in the second stage.

Algorithm 2: CRS Algorithm (1-m-1 Model)

```
Input  : A real-time composite task set $\mathcal{T} = \{\tau_i\}_{i=1}^n$
Output: A network schedule $\mathcal{S}_{\rm net}$ and a computing schedule
        $\mathcal{S}_{\rm com}$, if exist
1  Compute the hyper-period $\mathcal{H}$ and construct the set $I$ of
     instances of $\mathcal{T}$
2  while True do
3      $\mathcal{S}_{\rm com}$ = ComputingScheduling($I$)
4      if $\mathcal{S}_{\rm com}$ is infeasible then
5          return None
6      end
7      $\mathcal{S}_{\rm net}$ = NetworkScheduling($I, \mathcal{S}_{\rm com}$)
8      if $\mathcal{S}_{\rm net}$ is infeasible then
9          if ConstraintGenerator($I$) = None then
10             return None
11         end
12     else
13         break
14     end
15 end
16 return $\mathcal{S}_{\rm net}$, $\mathcal{S}_{\rm com}$
```

2) Constraint Generator: we first introduce some definitions and preliminaries to help understand the design of the constraint generator.

Definition 4. Virtual Release Time/Deadline: *Given a set of real-time composite tasks $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, for each instance $\tau_{i,j}$ with the release time $r_{i,j}$ and deadline $d_{i,j}$, the virtual release time and virtual deadline of the sensing segment $\tau_{i,j,s}$ is set to be $r_{i,j,s} = r_{i,j}$ and $d_{i,j,s} = s_{i,j,c}$. The virtual release time and deadline of the actuating segment $\tau_{i,j,a}$ is set to be $r_{i,j,a} = f_{i,j,c}$ and $d_{i,j,a} = d_{i,j}$, where $s_{i,j,c}$ and $f_{i,j,c}$ are the start time and the finish time of the corresponding computing segment $\tau_{i,j,c}$, respectively. Both $s_{i,j,c}$ and $f_{i,j,c}$ are obtained in the computing scheduling sub-problem.*

We say a network segment is *virtually included* in a time interval if its virtual release time and virtual deadline are both included in that interval. That is, a network segment $\tau_{i,j,\rm net}$ is virtually included in $[t_0, t_1]$ if $r_{i,j,\rm net} \geq t_0$ and $d_{i,j,\rm net} \leq t_1$. Based on this condition, we define a *virtual network demand* over a given interval to be the sum of the execution time of all the network segments that are virtually included in that interval. We define the *virtual network overload interval* and *minimal virtual network overload interval* as follows.

Definition 5. Virtual Network Overload Interval (VNOI): *Given a set of network segments and a time interval $[t_0, t_1]$, the interval $[t_0, t_1]$ is a virtual network overload interval if its virtual network demand is larger than $t_1 - t_0$.*

Definition 6. Min. Virtual Network Overload Interval: *Given a VNOI $[t_0, t_1]$, if there does not exist another VNOI $[t_2, t_3]$ satisfying $[t_2, t_3] \subset [t_0, t_1]$, $[t_0, t_1]$ is a minimal VNOI.*

For a VNOI $\alpha = [t_0, t_1]$, we use U_α to denote the set of network segments which are virtually included in α. We use $A_\alpha \subset U_\alpha$ to denote the set of actuating segments in U_α whose corresponding sensing segments are not virtually included in α. That is, for any $\tau_{i,a} \in A_\alpha$, $\tau_{i,a} \in U_\alpha$ and $\tau_{i,s} \notin U_\alpha$. We use $S_\alpha \subset U_\alpha$ to denote the set of sensing segments in U_α whose corresponding actuating segments are not virtually included in α. Finally, we use $M_\alpha \subset U_\alpha$ to denote the set of networking segments which contain both the sensing and actuating segments from the same tasks.

According to the definitions of U_α, A_α, S_α, and M_α, it holds that $U_\alpha = A_\alpha \cup S_\alpha \cup M_\alpha$. Let $A_{\alpha,c}$, $S_{\alpha,c}$ and $M_{\alpha,c}$ denote the sets of corresponding computing segments of the network segments in A_α, S_α and M_α, respectively. We define $\phi(A_{\alpha,c})$, $\phi(S_{\alpha,c})$, and $\phi(M_{\alpha,c})$ to be the amount of execution time of the computing segments in $A_{\alpha,c}$, $S_{\alpha,c}$, and $M_{\alpha,c}$ that are scheduled in the interval $[t_0 - 1, t_1 + 1]$ in the computing schedule.

It should be noted that there exists no feasible schedule for the network scheduling sub-problem when VNOIs are present. Thus, the constraint generator must eliminate all VNOIs by adding new constraints to the original CRS problem. In the following, we first present several important observations and then show how to eliminate a VNOI by adjusting the timing parameters of the network segments that are virtually included in that interval.

Lemma 4. *The virtual network demand over a VNOI $\alpha = [t_0, t_1]$ is at most $t_1 - t_0 + 2$ under the 1-m-1 model.*

Proof. Suppose the interval $\alpha = [t_0, t_1]$ has a total amount of $t_1 - t_0 + x$ units of network segments virtually included in $[t_0, t_1]$, where $x \in \mathbb{Z}_{>0}$. We will prove $x \leq 2$ as follows.

Based on Definition 4, the virtual release times of every two actuating segments are different since the finish times of their corresponding computing segments cannot be identical in the computing schedule. Thus, for any two different actuating segments $\tau_{i,j,a}, \tau_{x,y,a} \in U_\alpha$, it holds that $|r_{i,j,a} - r_{x,y,a}| \geq 1$. Similarly, the virtual deadlines of every two sensing segments are different since the start times of their corresponding computing segments cannot be identical in the computing schedule. Thus, for any two different sensing segments $\tau_{i,j,s}, \tau_{x,y,s} \in U_\alpha$, the condition $|d_{i,j,s} - d_{x,y,s}| \geq 1$ holds.

For any $\tau_{i,j,a} \in A_\alpha$, since its corresponding sensing segment is excluded from U_α, its corresponding computing segment $\tau_{i,j,c}$ has the release time $r_{i,j,c} \leq t_0$ and deadline $d_{i,j,c} \leq t_1 - 1$. The last unit of $\tau_{i,j,c}$ is scheduled in $[r_{i,j,a} - 1, r_{i,j,a}]$ in the computing schedule. Due to that $r_{i,j,a} \in [t_0, t_1 - 1]$, the last unit of $\tau_{i,j,c}$ can be scheduled in $[t_0 - 1, t_1 - 1]$. Taking into account all the computing segments in $A_{\alpha,c}$, it holds that $|A_\alpha| \leq \phi(A_{\alpha,c})$. Similarly, for any $\tau_{x,y,s} \in S_\alpha$, since its corresponding actuating segment is excluded from U_α, its corresponding computing $\tau_{x,y,c}$ has release time $r_{x,y,c} > t_0$ and deadline $d_{x,y,c} \geq t_1 + 1$. The first unit of $\tau_{x,y,c}$ is scheduled in $[d_{x,y,s} - 1, d_{x,y,s}]$ in the computing schedule. Due to that $d_{x,y,s} \in [t_0 + 1, t_1]$, at least the first unit of $\tau_{i,j,c}$ is scheduled in $[t_0 + 1, t_1 + 1]$. Considering all the computing segments in S_α, it holds that $|S_\alpha| \leq \phi(S_{\alpha,c})$. For any actuating segment $\tau_{u,v,a}$ in M_α, since its corresponding sensing segment and itself are both in the interval $[t_0, t_1]$, it holds that $r_{u,v,c} \geq t_0 + 1$ and $d_{u,v,c} \leq t_1 - 1$. So the computing segment $\tau_{u,v,c}$ is scheduled within $[t_0 + 1, t_1 - 1]$. Since $\tau_{u,v,c}$ has at least two units of execution time, it holds that $|M_\alpha| \leq \phi(M_{\alpha,c})$. Based on the above constraints on the timing parameters of the computing segments, it holds that

$$|S_\alpha| + |A_\alpha| + |M_\alpha| \leq \phi(S_{\alpha,c}) + \phi(A_{\alpha,c}) + \phi(M_{\alpha,c}) \quad (5)$$

As the amount of the execution time of the computing segments scheduled in the interval $[t_0 - 1, t_1 + 1]$ cannot exceed the length $t_1 - t_0 + 2$ of this interval, there is

$$\phi(S_{\alpha,c}) + \phi(A_{\alpha,c}) + \phi(M_{\alpha,c}) \leq t_1 - t_0 + 2 \quad (6)$$

Combining (5) and (6), it yields that

$$|S_\alpha| + |A_\alpha| + |M_\alpha| \leq t_1 - t_0 + 2 \quad (7)$$

Therefore, we conclude that $x \leq 2$ and the virtual network demand over an interval $[t_0, t_1]$ is at most $t_1 - t_0 + 2$ if the EDF schedule of the computing segments is feasible. □

Lemma 4 gives an upper bound on the virtual network demand over a virtual network overload interval. The number of overflow network segments in the interval is thus restricted to 2. However, selecting two feasible overflow network segments from a VNOI is still challenging due to its combinatorial nature. The following lemma presents an important property of the minimal VNOI (see Definition 6), which can further speed up the overflow segment selection from VNOIs.

Lemma 5. *The virtual network demand over a minimal VNOI $\alpha = [t_0, t_1]$ is at most $t_1 - t_0 + 1$ under the 1-m-1 model.*

Proof. Due to the page limit, please refer to the full technical report for the proof [28] □

Lemma 5 indicates that only one network segment needs to be moved out of a minimal VNOI. Based on this observation, we aim to select a network segment as the overflow segment and modify its timing parameters to prevent it from being scheduled in the interval. This procedure is defined as the elimination procedure. We first identify the *precondition* for a network segment to be selected as an overflow segment. That is, after the modification of the timing parameter(s) of its corresponding real-time composite task, the schedulability of the network scheduling sub-problem and computing scheduling sub-problem can still be preserved. Based on this precondition, we design the constraint generator based on a backtracking strategy to eliminate all VNOIs.

The elimination procedure for a minimal VNOI $\alpha = [t_0, t_1]$ is achieved based on a selected candidate segment $\tau_{i,j,\text{net}}$ and the instance set I. Consider that we select an actuating segment $\tau_{i,j,a} \in A_\alpha$. Let D be the set of timing parameters which include the deadlines of the actuating segments in A_α and t_0. After sorting D in the descending order, we can find $d_0 \in$

Algorithm 3: Constraint Generator (1-m-1 Model)

```
Input  : An instance set $I_0$ of real-time tasks
Output: A modified instance set $I_2$ or the initial instance set
        $I_0$
1  if the iterative two-stage decomposition method goes to
   network scheduling then
2      Construct the set $O$ of minimal VNOIs
3      if $O = \emptyset$ then
4          return $I_0$
5      end
6      Get the earliest minimal VNOI $\alpha \in O$
7      for $\tau_{i,\mathrm{net}} \in A_\alpha \cup S_\alpha$ do
8          $I_1$ = Eliminate($I_0, \alpha, \tau_{i,\mathrm{net}}$)
9          $I_2$ = ConstraintGenerator($I_1$)
10         if $I_2 \neq$ None then
11             return $I_2$
12         end
13     end
14 end
15 return None // Algorithm reports a failed case
```

D, the latest element smaller than $\bar{d}_{i,j,a}$. We modify deadline $\bar{d}_{i,j,a}$ to d_0 and $\bar{d}_{i,j,c}$ to $d_0 - 1$. This assures that the priority of $\tau_{i,j,c}$ improves, thus guaranteeing that their corresponding actuating segments will not be scheduled in α to make the interval become virtually overload again. After rescheduling the computing segments, the minimal VNOI α is eliminated.

On the other hand, if we select a sensing segment $\tau_{i,j,s} \in S_\alpha$, we obtain the set R of timing parameters including the release times of the sensing segments in S_α and t_1, and sort them in the ascending order. Let $r_0 \in R$ be the first element larger than $\bar{r}_{i,j,s}$, and we modify $\bar{r}_{i,j,s}$ to r_0. The minimal VNOI can be eliminated by rescheduling the computing segments and updating the virtual timing parameters.

Note that although any network segment virtually included in the interval $[t_0, t_1]$ can be taken as a candidate for the overflow segment, a candidate is infeasible if modifying its timing parameter hurts the schedulability of the real-time composite task set. Therefore, we design the constraint generator to eliminate VNOIs based on a backtracking algorithm.

Alg. 3 shows an overview of the constraint generator. The key idea is to utilize the backtracking search to eliminate all VNOIs through the constraint generator, where the input is the initial set of instances and the output is either a feasible instance set or reported failure. The constraint generator works as a recursion tree. The root corresponds to the original problem of finding a feasible instance set from the given instance set. Each node in this tree corresponds to a recursive sub-problem. In particular, when leaves cannot be further extended, it is either because the network scheduling sub-problem based on the current instance set finds infeasible schedules, or because the feasible instance set is found and returned. When the constraint generator is called, we construct the set O of minimal VNOIs based on the current instance set. If O is not empty, then starting with the earliest minimal VNOI in O, we branch the search based on its overflow candidate segments. For each candidate segment in the minimal VNOI, we call its elimination procedure to eliminate the overload interval. If every candidate segment fails to utilize the constraint generator to find an instance set which generates no VNOIs in the two-stage decomposition method, it indicates that the task set is not schedulable, which returns a None value and reports failure. The time complexity of the backtracking algorithm is $O(W^p N^2)$, where W is the average number of candidate segments in the minimal overload interval and p is the number of the overload intervals. When W approaches N, p goes to 1. On the contrary, when W approaches 1, p goes to N, where $N = \sum_{i=1}^{n} \mathcal{H}/T_i$ is the total amount of instances of all the tasks. Each node takes $O(N^2)$ time to identify the intervals.

Theorem 3. *If a feasible schedule exists for the real-time composite task set $\mathcal{T}$ under* 1-m-1 *model, Alg. 2 can find it.*

Proof. Due to the page limit, please refer to the full technical report for the proof [28] □

V. CRS Problem under General Model

Based on the studies of the two special cases, we now extend the CRS problem to its general case, where every real-time composite task has arbitrary execution time for each segment.

A. Algorithm Overview

We first extend the concepts of effective network overload/tight intervals (ENTI/ENOI, see Definition 2) to *effective computing overload/tight intervals*.

Definition 7. Effective computing overload/tight interval (ECOI/ECTI): *Given a set of real-time composite tasks and a time interval $[t_0, t_1]$, $[t_0, t_1]$ is an effective computing overload interval if the effective computing demand over $[t_0, t_1]$ is larger than $t_1 - t_0$. $[t_0, t_1]$ is an effective computing tight interval if the effective computing demand over $[t_0, t_1]$ is equal to $t_1 - t_0$.*

For the CRS problem in the general case, we first use EDF to schedule the network and computing segments in parallel based on their effective deadlines. Suppose EDF fails scheduling a segment and generates a partial schedule, we define the *provisional timing parameters* as follows.

Definition 8. Provisional Release Time/Deadline: *For each instance $\tau_{i,j}$ with release time $r_{i,j}$ and deadline $d_{i,j}$, the provisional release time $\hat{r}_{i,j,s}$ and provisional deadline $\hat{d}_{i,j,s}$ of its sensing segment $\tau_{i,j,s}$ are $\hat{r}_{i,j,s} = r_{i,j}$ and $\hat{d}_{i,j,s} = f_{i,j,s}$, where $f_{i,j,s}$ is the finish time of the sensing segment. The provisional release time $\hat{r}_{i,j,c}$ and provisional deadline $\hat{d}_{i,j,c}$ of its computing segment $\tau_{i,j,c}$ are $\hat{r}_{i,j,c} = f_{i,j,s}$ and $\hat{d}_{i,j,c} = f_{i,j,c}$, where $f_{i,j,c}$ is the finish time of the computing segment. The provisional release time $\hat{r}_{i,j,c}$ and provisional deadline $\hat{d}_{i,j,a}$ of its actuating segment $\tau_{i,j,a}$ are $\hat{r}_{i,j,a} = f_{i,j,c}$ and $\hat{d}_{i,j,a} = d_{i,j}$.*

The provisional timing parameters of each segment are initialized as its effective timing parameters before EDF is applied. We say a network/computing segment is *provisionally included* in a time interval if its provisional release time and provisional deadline are both within that interval. Based on this condition, we define a *provisional network/computing demand* over a given interval to be the sum of the execution time of all the network/computing segments that are provisionally included in that interval. We define the *provisional network/computing overload interval* as follows.

Definition 9. Provisional Network/Computing Overload Interval (PNOI/PCOI): *Given a set of network/computing segments and a time interval $[t_0, t_1]$, $[t_0, t_1]$ is a provisional network/computing overload interval if its provisional network/computing demand is larger than $t_1 - t_0$.*

For a PNOI $\alpha = [t_0, t_1]$, we use $\mathcal{X}_\alpha$ to denote the set of actuating segments provisionally included but not effectively included in $[t_0, t_1]$. Similarly, for a PCOI $\beta = [b_0, b_1]$, we use $\mathcal{C}_\beta$ to denote the set of computing segments that are provisionally included but not effectively included in $[b_0, b_1]$.

Alg. 4 gives an overview of the CRS algorithm under the general model. It is an iterative two-stage method utilizing two groups of intervals to modify the timing parameters of the tasks.

In stage 1, effective network/computing tight intervals (ENTI/ECTI, see Definition 2 and 7) are used to represent the intervals that have 100% utilization in computing and network resources in any feasible schedule, respectively. A network/computing segment with only one of its effective release time and effective deadline included in an ENTI/ECTI must be moved out of that interval. This adjustment provides better priority for those segments and prevents them from being scheduled in an ENTI/ECTI. After modifying the timing parameters, EDF is used to schedule the network and computing segments in parallel based on their effective deadlines. If EDF fails to find a feasible schedule, we turn to stage 2.

In stage 2, provisional network/computing overload intervals (PNOI/PCOI, see Definition 9), obtained by EDF scheduling are used to represent the intervals that have the provisional utilization larger than 100%. This indicates that we need to move some segments out of a PNOI/PCOI. Among all the candidate segments in a PNOI/PCOI, we propose a greedy strategy to select the actuating/computing segment with the earliest effective release time and move its effective deadline ahead. This will make its corresponding preceding segment(s), e.g., the sensing or/and computing segment, finish earlier, and create more space for scheduling the current segment.

Algorithm 4: CRS Algorithm (General Model)

```
Input  : A real-time composite task set $\mathcal{T} = \{\tau_i\}_{i=1}^n$
Output: A network schedule $\mathcal{S}_{\rm net}$ and a computing schedule
        $\mathcal{S}_{\rm com}$, if exist
1  Compute the hyper-period $\mathcal{H}$ and construct the set $I$ of
     instances of $\mathcal{T}$
2  while True do
       // Stage-1
3      if an ENOI/ECOI is identified then
4          return None // reports a failed case
5      end
6      if an identified ENTI/ECTI triggers the modification on
         timing parameters (Rule 1(a-b) and Rule 2(a-b)) then
7          continue
8      end
       // Stage-2
9      Run EDF to schedule the network and computing
         segments in parallel based on their effective deadlines
         to construct $\mathcal{S}_{\rm net}$ and $\mathcal{S}_{\rm com}$. EDF terminates if any
         segment misses the deadline. The provisional timing
         parameters are updated
10     if the earliest PCOI $\beta = [b_0, b_1]$ is located then
11         if $\mathcal{C}_\beta$ has no feasible candidate segment then
12             return None
13         end
14         Modify the candidate segment in $\mathcal{C}_\beta$ with earliest
             effective release time (Rule 3(a-b))
15         continue
16     end
17     if the earliest PNOI $\alpha = [t_0, t_1]$ is located then
18         if $\mathcal{X}_\alpha$ has no feasible candidate segment then
19             return None
20         end
21         Modify the actuating segment in $\mathcal{X}_\alpha$ with earliest
             effective release time (Rule 4(a-b))
22         continue
23     end
24     break
25 end
26 return $\mathcal{S}_{\rm net}$, $\mathcal{S}_{\rm com}$
```

B. Design details of the CRS algorithm under general model

We now present the details of the greedy heuristics.

1) Stage 1: Modification based on effective timing parameters: The first stage of the algorithm identifies ECOIs and ENOIs to decide the schedulability of the task set. It then utilizes ENTIs and ECTIs to adjust the tasks according to the following procedure.

Consider an ENTI $\alpha = [t_0, t_1]$, for any sensing/actuating segment $\tau_{i,j,s}$/$\tau_{i,j,a}$, we introduce Rule 1a and Rule 1b to modify its timing parameters, respectively.

- Rule 1a: if $\tau_{i,j,s}$ is not effectively included in α but has its effective release time $\overline{r}_{i,j,s}$ included in α, we modify $\overline{r}_{i,j,s}$ to t_1 and the effective release times of its computing and actuating segments are adjusted to $t_1 + C_{i,s}$ and $t_1 + C_{i,s} + C_{i,c}$, respectively.
- Rule 1b: if $\tau_{i,j,a}$ is not effectively included in α but has its effective deadline $\overline{d}_{i,j,a}$ included in α, we modify $\overline{d}_{i,j,a}$ to t_0 and the effective deadlines of its computing and sensing segments are adjusted to $t_0 - C_{i,a}$ and $t_0 - C_{i,a} - C_{i,c}$, respectively.

Similarly, for any ECTI $\alpha = [t_0, t_1]$, we introduce Rule 2a and Rule 2b to modify the timing parameters of a computing segment $\tau_{i,j,c}$:

- Rule 2a: if $\tau_{i,j,c}$ is not effectively included in α but has its effective release time $\overline{r}_{i,j,c}$ included in α, we modify $\overline{r}_{i,j,c}$ to t_1 and the release time of its actuating segment is adjusted to $t_1 + C_{i,c}$.
- Rule 2b: if $\tau_{i,j,c}$ is not effectively included in α but has its effective deadline $\overline{d}_{i,j,c}$ included in α, we modify $\overline{d}_{i,j,c}$ to t_0 and the effective deadline of its sensing segment is adjusted to $t_0 - C_{i,c}$.

To thoroughly check all the identified ECTIs and ENTIs and the newly generated ones due to the ongoing modifications, whenever a modification is made, we check again on all the ECTIs and ENTIs based on the modified task set until no more modifications is needed. In the meanwhile, we utilize ECOIs and ENOIs as the necessary conditions to decide the feasibility of the task set. If any ECOI or ENOI is found, the task set is unschedulable.

2) Stage 2: Modification based on provisional timing parameters: After stage 1, we use EDF to schedule the segments based on their effective deadlines. If EDF fails to construct a feasible schedule, the algorithm moves to the second stage to deal with PCOIs and PNOIs by employing greedy strategy that always modifies the candidate segment with the earliest effective release time.

Consider the first case that we locate the earliest PCOI $\beta = [b_0, b_1]$, where the provisional demand is $\mathcal{D}_\beta$ and the extra provisional demand satisfies $\mathcal{D}_{\beta,extra} = \mathcal{D}_\beta - b_1 + b_0$. Let $\mathcal{C}_\beta$ denote the set of computing segments included in β. We sort them by their effective release times in ascending order. When traversing $\mathcal{C}_\beta$, for a candidate computing segment $\tau_{i,j,c}$, if there exists a set of computing segments $\mathcal{C}_{\beta,1} \subset \mathcal{C}_\beta$ that have their effective deadlines smaller than $\overline{d}_{i,j,c}$, we introduce Rule 3a to modify $\tau_{i,j,c}$; if $\mathcal{C}_{\beta,1}$ does not exist, we introduce Rule 3b to modify $\tau_{i,j,c}$. If the modification generates no further ECOIs or ENOIs, we accept this modification and use EDF to schedule the segments again.

- Rule 3a: we modify the effective deadline $\overline{d}_{i,j,c}$ of $\tau_{i,j,c}$ to the effective deadline $\overline{d}_{x,y,c}$ of another computing segment $\tau_{x,y,c}$ in $\mathcal{C}_{\beta,1}$. $\overline{d}_{x,y,c}$ is the latest one among all the computing segments in $\mathcal{C}_{\beta,1}$.
- Rule 3b: if $\mathcal{D}_{\beta,extra} \geq C_{i,c}$, we modify $\overline{d}_{i,j,c}$ to b_0. If $\mathcal{D}_{\beta,extra} < C_{i,c}$, we modify $\overline{d}_{i,j,c}$ to $b_0 + C_{i,c} - \mathcal{D}_{\beta,extra}$.

We further consider the second case where the earliest PNOI $\alpha = [t_0, t_1]$ is found. Its provisional demand is $\mathcal{D}_\alpha$ and the extra provisional demand is $\mathcal{D}_{\alpha,extra} = \mathcal{D}_\alpha - t_1 + t_0$. After sorting the set of actuating segments in $\mathcal{X}_\alpha$ by their effective release times in ascending order, we traverse $\mathcal{X}_\alpha$. For a candidate actuating segment $\tau_{i,j,a}$, if there exists a set of actuating segments $\mathcal{X}_{\alpha,1} \subset \mathcal{X}_\alpha$ that have their effective deadlines smaller than $\overline{d}_{i,j,a}$, we introduce Rule 4a to modify $\tau_{i,j,a}$; if $\mathcal{X}_{\alpha,1}$ does not exist, we introduce Rule 4b to modify $\tau_{i,j,a}$.

- Rule 4a: we modify the effective deadline $\overline{d}_{i,j,a}$ to the effective deadline of another actuating segment $\tau_{x,y,a}$ in the set $\mathcal{X}_{\alpha,1}$ which has the latest effective deadline.
- Rule 4b: if $\mathcal{D}_{\alpha,extra} \geq C_{i,a}$, we modify $\overline{d}_{i,j,a}$ to t_0. If $\mathcal{D}_{\alpha,extra} < C_{i,a}$, we modify $\overline{d}_{i,j,a}$ to $t_0 + C_{i,a} - \mathcal{D}_{\alpha,extra}$.

Modifying the effective deadline of the actuating segment will change the effective deadlines of its corresponding computing and sensing segments. If the modification of a candidate actuating segment incurs no ECOIs or EVOIs, the modification is accepted. If there exists no feasible candidate segment, the algorithm returns failure.

The key design principle of the above greedy heuristics is based on the observation that a segment with an earlier effective release time is usually preempted by the segments with later effective release times. The proposed adjustment improves the priority of the current candidate segment. In addition, since the effective release time of its preceding segment(s) is also modified, it makes more space for scheduling the current candidate segment. The segment with the earliest effective release time is preempted by the largest amount of other segments, thus the modification enables the effective adjustment of the priority to the utmost. This modification procedure repeats until EDF finds feasible schedules. Each segment can be modified at most $O(N)$ time and there are at most $O(N)$ number of segments. Since for each round of modification, identifying the overload/tight intervals takes $O(N^2)$ time, the overall time complexity of Alg. 4 is $O(N^4)$, where $N = \sum_{i=1}^{n} \mathcal{H}/T_i$ is the total amount of instances of all the tasks.

VI. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms for solving the CRS problem under the h-1-1, 1-m-1, and the general model. The proposed CRS algorithms are compared with the baseline algorithms, including both EDF and least laxity first (LLF). For EDF, we use two ready queues to separately store the network and computing segments. A segment is popped out from the queue if its corresponding task has the earliest deadline. A segment is released and pushed into the ready queue when its preceding segment is finished. Similarly, LLF also uses two ready queues. In each queue, the priority of the segments is defined as the laxity of the tasks, i.e. the task deadline minus the remaining execution time. We use NEC to denote the number of task set satisfying the necessary condition of finding a feasible schedule, which is the upper bound of the feasible cases under the general model. We search ECOI and ENOI for each trial of task set. If any ECOI or ENOI is identified, the trial will be recorded as an infeasible case and will not be counted in NEC.

A. Experimental Setup

To efficiently obtain the feasible solution of constraint programming, we employ an efficient satisfiability modulo theories (SMT) solver Z3. All the algorithms including SMT are implemented in Python and computed in a CPU cluster node with Xeon E5-2690 v3 2.6 GHz CPU. To perform an extensive comparison, we generate 1000 trials under each parameter setting. Each trial contains a task set $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$, where $n \in [1, 50]$. The task periods are randomly set in $[10, 10000]$. For the task set generated under the h-1-1 model, since its network resource utilization is much larger than the computing resource utilization, we vary the network resource utilization to assess the impact. Similarly, we vary the computing resource utilization of the task set generated under the 1-m-1 model. For the task set under the general model, we use the normalized resource utilization to represent the resource utilization of a task, where the normalized utilization is calculated as $(C_{i,s} + C_{i,c} + C_{i,a})/2T_i$ (we use $2T_i$ because there are two resources, CPU and network bandwidth). Given the utilization of a task set, we generate the utilization of all the tasks using the UUniSort algorithm [29]. For the general model, we randomly split the total execution time into the sensing, computing, and actuating segments.

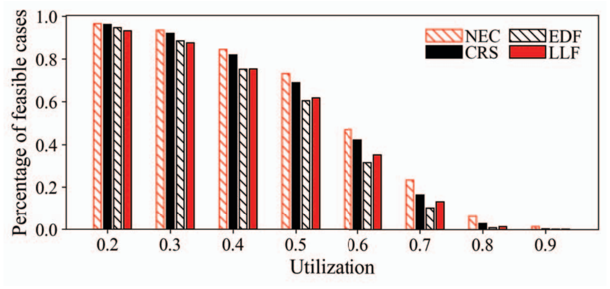
B. Evaluation Results

In the first set of experiments, we compare the performance of the proposed solutions with the baseline methods for the general model, h-1-1 model, and 1-m-1 model by varying the resource utilization of the task set. Fig. 3a shows the percentage of feasible cases obtained by CRS, EDF, and LLF under the general model when the normalized resource utilization is varied. One can observe that as the normalized resource utilization increases from 0.2 to 0.9, the percentage of feasible cases that NEC, CRS, EDF, and LLF can achieve gradually decrease from 96.6%, 96.2%, 94.7%, and 93.2% to 1.3%, 0.3%, 0.1%, and 0.1%, respectively. Fig. 3a also shows that the performance of the proposal heuristic solution is close to the upper bound (the difference increases from 0.4% to 7% when the utilization is increased from 0.2 to 0.9), and outperforms those of EDF and LLF significantly. For example, it reports more than 10% and 7% feasible cases than EDF and LLF, respectively, when the utilization is set at 0.6.

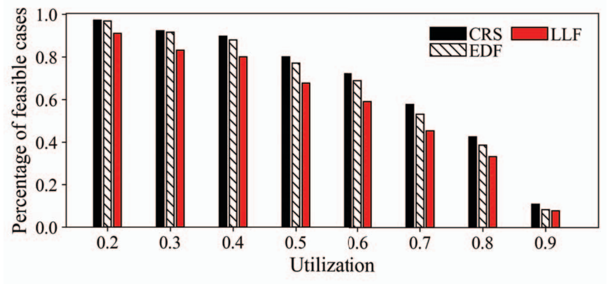
The performance of the algorithms under h-1-1 model is shown in Fig. 3b. It is observed that the percentage of feasible cases that CRS, EDF, and LLF can achieve gradually decrease from 97.3%, 96.8%, and 91.0% to 10.9%, 8.4%, and 7.8%, respectively. Our method outperforms EDF and LLF when the utilization is high. For example, it reports 4.5% and 12% more feasible cases than EDF and LLF, respectively, when the utilization is set at 0.7. We also evaluate the performance of the proposed algorithm by varying the computing resource utilization of the task set under 1-m-1 model. Fig. 3c reports the percentage of feasible schedules obtained by CRS, EDF, and LLF, which decrease from 97.5%, 97.5%, and 89.9% to 20.0%, 18.6%, and 15.6%, respectively. CRS shows slightly better performance than EDF due to a small number of cases that identify the virtual network overload intervals. Note that we don't show the upper bound (NEC) for h-1-1 model and 1-m-1 model because the proposed method is optimal.

In the second set of experiments, we further evaluate the performance of the SMT method for the general model. The results are shown in Table II. In the experiments, we generate 1000 cases for the evaluation. Since SMT may take a long time to obtain a solution in some corner cases, we set the longest task period to 1000s in each case and the average number of jobs is 100 for the 1000 cases. We also set a time limit of 2 hours for running SMT. When the time limit is reached, SMT terminates and that trial will be recorded as an unsolved case. From Table II, it is observed that the average running time of SMT for the solved cases is 1940s and the percentage of solved cases of SMT is only 15.1%. Compared with SMT, our method can finish a trial in 74 milliseconds on average and can solve all the 1000 cases.

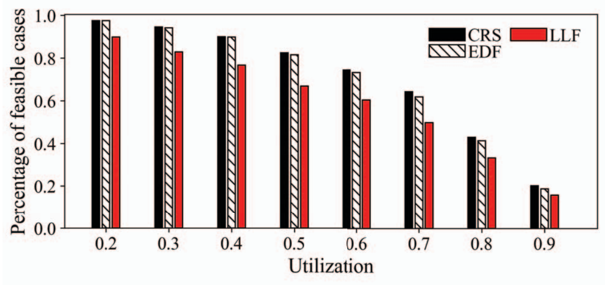
In the last set of experiments, we evaluate the running time of the algorithms by increasing the number of jobs from 100 to 10000. The comparison results are shown in Fig. 4. It can be observed that the average running time of all the algorithms increase along with the increase of the number of jobs. For all the models, the running time of CRS are under 20s when the number of jobs are less than 2000. When the number of jobs reaches 10,000, the running time of CRS is under 500s, which is acceptable compared with the running time of EDF and LLF methods. SMT terminates due to the time limit (7200s) when the number of jobs are larger than 1000. Note in Fig. 4, we only show the running time of the feasible cases for CRS, EDF, and LLF. For the infeasible cases, the average running time is under 10s.



(a) General model



(b) h-1-1 model



(c) 1-m-1 model

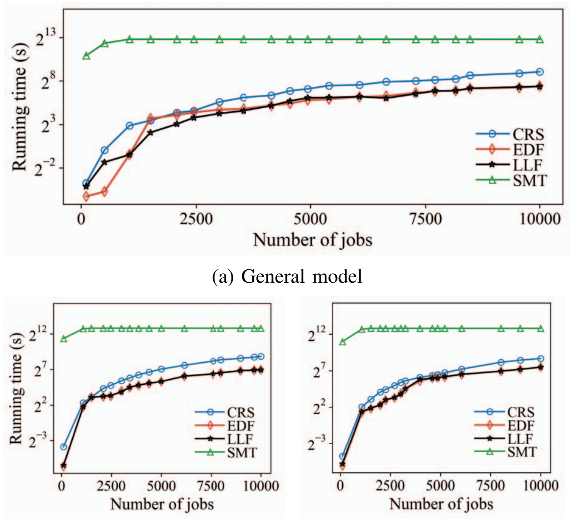
Fig. 3: Percentage of feasible cases versus utilization under different models

TABLE II: Performance comparison with SMT

	Average running time		Percentage of terminated cases due to time limit
	Unsolved	Solved	
SMT	>2h	1940s	84.9%
CRS	N/A	0.074s	0%
EDF	N/A	0.008s	0%
LLF	N/A	0.008s	0%

VII. Related Work

Most real-time networks adopt Time Division Multiple Access (TDMA) based data link layers to guarantee deterministic real-time communications. Sensing and actuating tasks are abstracted as end-to-end flows with specified timing requirements. The existing real-time network scheduling algorithm



(a) General model

(b) h-1-1 model

(c) 1-m-1 model

Fig. 4: Comparison of running time with different number of jobs

designs focus on schedulability analysis and management of the network packet scheduling (e.g., [8], [30]–[34]). Those solutions may fit well for NCSs when the transmission time is large and the computation time of the controller is negligible. However, with the rapid development of real-time network such as RT-WiFi [35], which supports a minimal time slot of 0.1 milliseconds, the transmission time of the network segment becomes comparable to the computation time of controller [20]. In addition, due to an increasing number of industrial applications having complex online optimization algorithms designed for the controllers, the computation time can no longer be ignored [15], [17]. In this paper we propose a three-segment execution model for scheduling composite network and CPU resources in NCSs. To provide more flexibility, both network segments and computing segments are considered to be preemptive.

The existing related three-segment execution models include self-suspension model, PRedictable Execution Model (PREM) and Acquisition-Execution-Restitution (AER) model. The self-suspension model is composed of two computation segments separated by one suspension interval, which focuses on one resource type [36]. PREM and AER are designed for multi-core CPU system. PREM enables parallelism by dividing tasks in communication/computation phases [22]. The read phase reads data from the main memory, the computation phase can proceed the execution, and the write phase writes the resulting data back to the main memory. Since PREM increases the predictability of an application by isolating memory accesses, it is widely used [37]–[40]. However, the PREM model usually couples the write phase of a task with the next activated read phase on the same core [39], [41]. By contrast, AER from [24] allows more freedom to schedule the read and write phases, which is further implemented on a scratchpad memory (SPM) based single-core [39]. Hiding the communication latency when scheduling a task graph on a multi-core based on the AER model is discussed in [42], [43]. Other research works based on the AER model studies how to improve the schedulability of latency-sensitive tasks, manage the memory on COTS multiprocessor platforms, and increase system determinism by reducing task switching overhead, and the global static scheduling of non-preemptive tasks [44]–[47]. As those recent works above focus on scheduling for multi-core CPU systems, the communication phases are reasonably considered to be non-preemptive, while the network scheduling in our CRS model employs the preemptive segments to provide better parallelism.

For the existing related works on end-to-end scheduling in NCSs, utilizing the system state of control systems to design the scheduler is an important methodology to optimize QoS. Based on the system state, the scheduling and feedback co-design for NCSs is introduced in [48], [49], which computes the deadline for the real-time tasks. However, the network scheduling is not guaranteed to be real-time because of the CAN protocol considered in the model. The work in [50] studies how to integrate security guarantees with end-to-end timeliness requirements for control tasks in resource-constrained NCSs. The proposed sensing-control-actuation model is similar to our CRS model, but the sensing, computing and actuating segments in the proposed model have designed release times and deadlines. This is not general as the CRS model which only has a task deadline. Other related network and computing co-scheduling works include the co-generation of static network and task schedules for distributed systems which only focuses on the SMT solver [51], and the feasible time-triggered schedule configuration for control applications, which aims at minimizing the control performance degradation of the applications [52].

VIII. Conclusion and Future Work

In this paper, we study the composite resource scheduling problem in networked control systems (NCSs). A new composite resource scheduling (CRS) model is introduced to describe the sensing, computing, and actuating segments in NCSs. We formulate the composite resource scheduling problem in constraint programming and prove it to be NP-hard in the strong sense. Two special models and the general model are studied. For the CRS problem under the h-1-1 model, we present an optimal algorithm that utilizes the intervals of network resource utilization of 100% to prune the search space to find the solution. For the CRS problem under the 1-m-1 model, we propose an optimal algorithm that exploits a novel backtracking strategy to adjust the timing parameters of the tasks so that there exist no intervals of network resource utilization larger than 100% obtained in the two-stage decomposition method. For the general case, we propose a heuristic solution to find the feasible schedule based on the greedy strategy of modifying the segments in the interval of either network resource utilization or computing resource utilization larger than 100%.

As the future work, the proposed algorithms will be implemented on our NCS testbed to evaluate their effectiveness and practicability in real-life systems.

IX. ACKNOWLEDGEMENT

This work was partly supported by the National Science Foundation under NSF Award CNS-2008463 and the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project UGC/FDS11/E03/19.

REFERENCES

[1] X. Hei, X. Du, S. Lin, and I. Lee, "PIPAC: Patient infusion pattern based access control scheme for wireless insulin pump system," in *2013 Proceedings IEEE INFOCOM*, Apr. 2013, pp. 3030–3038.

[2] K. Gatsis, A. Ribeiro, and G. J. Pappas, "Optimal Power Management in Wireless Control Systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 6, pp. 1495–1510, Jun. 2014.

[3] G. Tian, S. Camtepe, and Y.-C. Tian, "A Deadline-Constrained 802.11 MAC Protocol With QoS Differentiation for Soft Real-Time Control," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 2, pp. 544–554, Apr. 2016.

[4] D. P. Borgers, R. Geiselhart, and W. P. M. H. Heemels, "Tradeoffs between quality-of-control and quality-of-service in large-scale nonlinear networked control systems," *Nonlinear Analysis: Hybrid Systems*, vol. 23, pp. 142–165, Feb. 2017.

[5] X.-S. Zhan, X.-X. Sun, J. Wu, and T. Han, "Optimal modified tracking performance for networked control systems with QoS constraint," *ISA Transactions*, vol. 65, pp. 109–115, Nov. 2016.

[6] S. Han, X. Zhu, A. K. Mok, D. Chen, and M. Nixon, "Reliable and Real-Time Communication in Industrial Wireless Mesh Networks," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2011, pp. 3–12.

[7] Q. Leng, Y. Wei, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "Improving Control Performance by Minimizing Jitter in RT-WiFi Networks," in *2014 IEEE Real-Time Systems Symposium*, Dec. 2014, pp. 63–73.

[8] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-Time Scheduling for WirelessHART Networks," in *2010 31st IEEE Real-Time Systems Symposium*, Nov. 2010, pp. 150–159.

[9] 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, May 2016.

[10] D. Yang, Y. Xu, H. Wang, T. Zheng, H. Zhang, H. Zhang, and M. Gidlund, "Assignment of Segmented Slots Enabling Reliable Real-Time Transmission in Industrial Wireless Sensor Networks," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3966–3977, Jun. 2015.

[11] B. Dezfouli, M. Radi, and O. Chipara, "Mobility-aware real-time scheduling for low-power wireless networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[12] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. Brest, France: Association for Computing Machinery, Oct. 2016, pp. 183–192.

[13] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive Software-defined Network (TSSDN) for Real-time Applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. Brest, France: Association for Computing Machinery, Oct. 2016, pp. 193–202.

[14] C. Lin, J. Zhou, C. Guo, H. Song, G. Wu, and M. S. Obaidat, "TSCA: A Temporal-Spatial Real-Time Charging Scheduling Algorithm for On-Demand Architecture in Wireless Rechargeable Sensor Networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 211–224, Jan. 2018.

[15] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of Autonomous Car—Part I: Distributed System Architecture and Development Process," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 12, pp. 7131–7140, Dec. 2014.

[16] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "Optimizing Model Predictive Control horizons using Genetic Algorithm for Motion Cueing Algorithm," *Expert Systems with Applications*, vol. 92, pp. 73–81, Feb. 2018.

[17] K. Jo, J. Kim, D. Kim, C. Jang, and M. Sunwoo, "Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 8, pp. 5119–5132, Aug. 2015.

[18] S. Klus, F. Nüske, S. Peitz, J.-H. Niemann, C. Clementi, and C. Schütte, "Data-driven approximation of the Koopman generator: Model reduction, system identification, and control," *Physica D: Nonlinear Phenomena*, vol. 406, p. 132416, May 2020.

[19] K.-H. Lee, D. K. Fu, M. C. Leong, M. Chow, H.-C. Fu, K. Althoefer, K. Y. Sze, C.-K. Yeung, and K.-W. Kwok, "Nonparametric Online Learning Control for Soft Continuum Robot: An Enabling Technique for Effective Endoscopic Navigation," *Soft Robotics*, vol. 4, no. 4, pp. 324–337, Aug. 2017.

[20] W. Chen, P. Wu, P. Huang, A. K. Mok, and S. Han, "Online Reconfiguration of Regularity-Based Resource Partitions in Cyber-Physical Systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2019, pp. 495–507.

[21] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A Predictable Execution Model for COTS-Based Embedded Systems," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2011, pp. 269–279.

[22] A. Alhammad and R. Pellizzoni, "Time-predictable execution of multithreaded applications on multicore systems," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–6.

[23] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo, "Global Real-Time Memory-Centric Scheduling for Multicore Systems," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2739–2751, Sep. 2016.

[24] C. Maia, L. Nogueira, L. M. Pinho, and D. G. Pérez, "A closer look into the AER Model," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sep. 2016, pp. 1–8.

[25] I. Griva, S. G. Nash, and A. Sofer, *Linear and Nonlinear Optimization*, 2nd ed. USA: SIAM, Jan. 2009.

[26] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[27] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[28] P. Wu, C. Fu, T. Wang, M. Li, Y. Zhao, C. J. Xue, and S. Han, "Composite Resource Scheduling for Networked Control Systems," Sep. 2021. [Online]. Available: https://arxiv.org/abs/2109.13211v1

[29] E. Bini and G. C. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, no. 1, pp. 129–154, May 2005.

[30] 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, Nov. 2018, conference Name: IEEE Transactions on Industrial Informatics.

[31] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "End-to-End Communication Delay Analysis in Industrial Wireless Networks," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1361–1374, May 2015, conference Name: IEEE Transactions on Computers.

[32] 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)*, Apr. 2017, pp. 261–272.

[33] 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)*, 2019.

[34] T. Zhang, T. Gong, S. Han, Q. Deng, and X. S. Hu, "Fully Distributed Packet Scheduling Framework for Handling Disturbances in Lossy Real-Time Wireless Networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 502–518, Feb. 2021.

[35] Y. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, "RT-WiFi: Real-Time High-Speed Communication Protocol for Wireless Cyber-Physical Control Applications," in *2013 IEEE 34th Real-Time Systems Symposium*, Dec. 2013, pp. 140–149, iSSN: 1052-8725.

[36] J.-J. Chen, T. Hahn, R. Hoeksma, and N. Megow, "Scheduling Self-Suspending Tasks: New and Old Results," p. 23, 2019.

[37] M. Becker, D. Dasari, B. Nicolic, B. Akesson, V. Nélis, and T. Nolte, "Contention-Free Execution of Automotive Applications on a Clustered

Many-Core Platform," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, Jul. 2016, pp. 14–24, iSSN: 2159-3833.

[38] M. Becker, S. Mubeen, D. Dasari, M. Behnam, and T. Nolte, "Scheduling multi-rate real-time applications on clustered many-core architectures with memory constraints," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2018, pp. 560–567, iSSN: 2153-697X.

[39] S. Wasly and R. Pellizzoni, "Hiding memory latency using fixed priority scheduling," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2014, pp. 75–86, iSSN: 1545-3421.

[40] G. Durrieu, M. Faugère, S. Girbal, D. Gracia Pérez, C. Pagetti, and W. Puffitsch, "Predictable Flight Management System Implementation on a Multicore Processor," in *Embedded Real Time Software (ERTS'14)*, TOULOUSE, France, Feb. 2014.

[41] A. Alhammad, S. Wasly, and R. Pellizzoni, "Memory efficient global scheduling of real-time tasks," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, Apr. 2015, pp. 285–296, iSSN: 1545-3421.

[42] M. R. Soliman and R. Pellizzoni, "WCET-Driven Dynamic Data Scratchpad Management With Compiler-Directed Prefetching," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Bertogna, Ed., vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 24:1–24:23, iSSN: 1868-8969.

[43] B. Rouxel, S. Skalistis, S. Derrien, and I. Puaut, "Hiding Communication Delays in Contention-Free Execution for SPM-Based Multi-Core Architectures," in *ECRTS 2019 - 31st Euromicro Conference on Real-Time Systems*, Stuttgart, Germany, Jul. 2019, pp. 1–24.

[44] D. Casini, P. Pazzaglia, A. Biondi, M. D. Natale, and G. Buttazzo, "Predictable Memory-CPU Co-Scheduling with Support for Latency-Sensitive Tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, Jul. 2020, pp. 1–6, iSSN: 0738-100X.

[45] G. Schwäricke, T. Kloda, G. Gracioli, M. Bertogna, and M. Caccamo, "Fixed-Priority Memory-Centric Scheduler for COTS-based Multiprocessors," Jul. 2020.

[46] R. Koike, T. Fukunaga, S. Igarashi, and T. Azumi, "Contention-Free Scheduling for Clustered Many-Core Platform," in *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, Dec. 2020, pp. 1–8.

[47] M. Schuh, C. Maiza, J. Goossens, P. Raymond, and B. D. d. Dinechin, "A study of predictable execution models implementation for industrial data-flow applications on a multi-core platform with shared banked memory," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2020, pp. 283–295, iSSN: 2576-3172.

[48] M. Branicky, S. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 2, Dec. 2002, pp. 1211–1217 vol.2, iSSN: 0191-2216.

[49] I. Saha, S. Baruah, and R. Majumdar, "Dynamic scheduling for networked control systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '15. New York, NY, USA: Association for Computing Machinery, Apr. 2015, pp. 98–107.

[50] V. Lesi, I. Jovanov, and M. Pajic, "Integrating Security in Resource-Constrained Cyber-Physical Systems," *ACM Transactions on Cyber-Physical Systems*, vol. 4, no. 3, pp. 28:1–28:27, May 2020.

[51] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, Mar. 2016.

[52] A. Minaeva, D. Roy, B. Akesson, Z. Hanzalek, and S. Chakraborty, "Control Performance Optimization for Application Integration on Automotive Architectures," *IEEE Transactions on Computers*, pp. 1–1, 2020.