IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 9, 2021, accepted May 24, 2021, date of publication June 4, 2021, date of current version June 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3086304

FT-PIP: Flush Task Incorporated Priority-Inheritance Protocol to Reduce Information Leakage on Multiprocessor Real-Time Systems

HYEONGBOO BAEK[1] AND JINKYU LEE[2], (Senior Member, IEEE)
[1]Department of Computer Science and Engineering, Incheon National University, Incheon 22012, South Korea
[2]Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Jinkyu Lee (jinkyu.lee@skku.edu)

This work was supported in part by the Incheon National University Research Grant in 2020, and in part by the National Research Foundation of Korea under Grant NRF-2019R1F1A1059663, Grant NRF-2021R1A2B5B02001758, and Grant NRF-2017H1D8A2031628.

ABSTRACT In a traditional real-time system, the major requirement is to finish every real-time task within its predefined time. However, as a modern real-time system is connected to external networks and its subsystems are developed by different vendors, it becomes an important requirement to address the problem of information leakage on resources shared by different real-time tasks. This triggers studies that incorporate security mechanisms into the real-time scheduling. While it is one of the most well-known approaches to add the flush task (FT) mechanism and analyze its effect on timing guarantees, the existing FT approaches have been limited to a real-time system that employs at most one resource shared by real-time tasks executed on a uniprocessor platform. In this paper, we propose a flush task incorporated priority-inheritance protocol (FT-PIP) for global fixed-priority multiprocessor scheduling and develop its schedulability analysis, which is the first attempt that not only (a) supports timing guarantees but also (b) satisfies security constraints for (c) a multiprocessor platform with multiple shared resources. Via simulations, we demonstrate that FT-PIP and its schedulability analysis are effective in achieving both (a) and (b) for (c).

INDEX TERMS Real-time scheduling, multiprocessor systems, flush task incorporated priority-inheritance protocol, information leakage.

I. INTRODUCTION

A system is referred to as a *real-time* system, if it is required to be correct in terms of both functionality and timing [1]. An unmanned reconnaissance aerial vehicle is a compelling example of a real-time system; it periodically conducts designated tasks such as receiving commands from a ground station, observing the target terrain, and sending the observation result back to the ground station [2]. Such real-time tasks should be not only performed in a functionally correct manner, but also completed until a predefined time instant called *deadline*. The latter is referred to as a real-time constraint, and it is a fundamental issue in designing real-time systems to satisfy the real-time constraints with limited computing resources (e.g., CPU, cache, DRAM, I/O interconnections, etc.). To satisfy the timing constraints for various domains of real-time systems, the real-time system community has tried to develop a methodology how to effectively assign computing resources to real-time tasks (called real-time scheduling algorithm) and a mathematical method to guarantee whether every real-time task can be completed in its corresponding deadline (called schedulability analysis) [3].

While conventional studies on real-time systems have mostly focused on the real-time constraints only, recent studies are trying to incorporate a security constraint into real-time scheduling algorithms and schedulability analyses [4]–[9]. Information leakage of the traditional real-time systems was not considered as an important issue compared to general purpose systems because traditional real-time systems normally operated in a restricted environment (isolated from the outside world) with specialized network protocols. However, there is an increasing security attacks on modern real-time systems whose subsystems developed by different

The associate editor coordinating the review of this manuscript and approving it for publication was Xiali Hei.

vendors are integrated and/or connected with insecure networks. The information leakage problem can often arise in such integrated systems as some different subsystems share the same computing resources on which sensitive data (e.g., private keys) are gleaned by adversaries via security attacks (e.g., side-channel attacks) [4].

Among several existing approaches to address the security problem for real-time systems, the flush task (FT) is a well-known mechanism to prevent the information leakage on shared resources by conditionally initiating the state of shared resources before each task is scheduled [5]–[7]. Since such a security mechanism inevitably causes additional timing overhead, real-time scheduling algorithms and their schedulability analysis should be adapted. To cope with this issue, Mohan *et al.* first proposed a fixed-priority (FP) non-preemptive scheduling algorithm that incorporates the FT mechanism and developed its corresponding schedulability analysis on a uniprocessor platform, which is based on the task model that enables each task to have a different security level [6]. The key technique is to upper-bound the total timing cost of the invoked flush tasks and incorporate it into the proposed schedulability analysis method. This work was extended by Pellizzoni *et al.* to a preemptive scheduling algorithm and a more general task model referred to as the *noleak* model [5]. The main shortage of these methods is limited practicality. This is because, they are only applicable to a uniprocessor platform for a single type of shared computing resources without considering mutual exclusion in a critical section of shared resources although modern real-time systems are normally equipped with multiple processors and shared resources.

It is well-known that write-operations in a critical section of a stateful shared resource (e.g., DRAM) conducted by concurrent accesses of different tasks can cause an undesired result called data inconsistency. Thus, synchronized mechanisms such as semaphore should be effectively utilized to prevent the data inconsistency. However, such synchronized mechanisms cause the priority-inversion problem that leads to the unexpected further waiting time for higher-priority tasks [10], [11].

As a solution to address such a priority inversion problem, resource-locking protocols (also known as resource-sharing protocols) have been extensively studied; the studies dynamically reallocate the priority of tasks to effectively utilize limited computing resources [10], [12], [13]. The priority-inheritance protocol (PIP) and priority ceiling protocol (PCP) [10] are the most popular ones, which promote the priority of tasks to the predefined level to avoid the priority inversion problem on a uniprocessor platform. The multiprocessor priority-inheritance protocol (MPIP) [12] and the distributed priority ceiling protocol (DPCP) [13] are variants for a multiprocessor platform. However, these protocols do not consider the security-related constraints although the information leakage problem should be considered for modern real-time systems.

In this paper, we aim at satisfying both (i) real-time and (ii) security constraints for multiprocessor platforms with (iii) mutual exclusion in every critical section for multiple shared resources. To this end, we propose the flush task incorporated priority-inheritance protocol (FT-PIP) for global FP multiprocessor scheduling, and the new response-time analysis (RTA) [15] for the FP scheduling under FT-PIP. Then, we achieve (i) as the proposed RTA guarantees the schedulability of our scheduling framework. Also, (ii) is realized by incorporating FT into FP scheduling where multiple types of resources are conditionally flushed in accordance with the predefined *noleak* model. We then guarantee (iii) by additionally applying PIP to the proposed framework for both (i) and (ii). Among a number of security threats for real-time systems, we focus on information leakage caused by the cache-based side channel attack that will be detailed in Section II-A.

The key challenge to achieve (i), (ii) and (iii) simultaneously is as follows. First, the scheduling pattern is highly sophisticated due to conditionally-invoked FTs (regarding (ii)) for multiple resources and dynamically-changing task priorities as well as waiting time to enter a critical section (regarding (iii)). Second, the proposed RTA should be able to judge whether there is no deadline miss for every task on the scheduling pattern (regarding (i)). To address the challenge, we upper-bound the number of FTs invoked during the execution of each task by transforming it to the problem of max-flow for each resource type. We then develop the new RTA that derives upper bounds of timing overhead stemming from all factors of (ii) and (iii).

This paper makes the following contributions.

- We propose a new *noleak* model that considers multiple types of shared resources to satisfy security constraints.
- We propose FT-PIP that incorporates the FT mechanism in accordance with the new *noleak* model into the multiprocessor real-time scheduling in which mutual execution with multiple types of shared resources is guaranteed under PIP.
- We develop a new RTA to guarantee the schedulability of real-time tasks under the proposed scheduling framework.

II. ADVERSARY AND SYSTEM MODEL

In this section, we first present our adversary model that describes assumptions for the considered attacker's capability and attack scenario. Then, we present our system model.

A. ADVERSARY MODEL

As we follow the common assumptions of the existing studies for real-time systems that consider the timing-inference attacks to induce information leakage on shared resources [4], [6], [29], [31], we also assume that an adversary knows the task parameters of a target task (called victim task) for his security attack and aims at gleaning the sensitive data on the resources shared by different tasks. This assumption is based on the fact that many critical functions of a real-time

control systems are normally conducted periodically, and its operation period can be deducted by observing its physical behavior [4], [29]. Also, the attacker will investigate the design and system details before attacking the target systems.

Among a number of timing-inference attacks exploiting the deterministic schedules of real-time systems, we take the cache-based side channel attack for an example described as follows [32], [33]. In the considered environment, cache sets are associated with the main memory, a task (called attacker task denoted by $\tau_A \in \tau$) consecutively executed with a victim task (denoted by $\tau_B \in \tau$) is hijacked by the attacker. That is, the execution pattern of $\tau_A \rightarrow \tau_B \rightarrow \tau_A$ is exhibited repetitively, and τ_B runs a crypto algorithm with sensitive data such as a private key stored in the main memory. The attacker tries to infer the address in the main memory at which the private key is located by repeating the following three operations:

- PRIME: τ_A fills some (or all) cache sets with its own data before τ_B performs its execution;
- IDLE: after τ_A yields (or completes) its execution, it waits for a certain time while τ_B executes; and
- PROBE: τ_A conducts the same operation as PRIME and measures the time to load its data for all cache sets.

During the PROBE operation, if τ_B accessed some cache sets, the cache sets will be replaced by the τ_B's data, and then τ_A observes the access time larger (due to cache misses) than that of the other cache sets inducing cache hits. τ_A repetitively collects such the timing information and infers the location of the private key in the main memory.

The success of the above attack is dependent on which task can be hijacked by the attacker and how narrow the time range τ_B can appear. To this end, the attacker first obtains the exact schedule by observing busy periods deduced by the idle task when tasks are scheduled by the FP scheduling [28]. When τ_A executes frequently and its priority is higher than that of τ_B, it is more advantageous for the attacker to collect more information within a short period of time. Our interest is to prevent such a timing inference attack by conditionally initiating the status of shared resources when such initiation is needed according to the associated noleak table for the considered real-time tasks.

B. SYSTEM MODEL

We consider a set of n sporadic tasks $\tau = \{\tau_1, \cdots, \tau_n\}$, each of whose characteristic is specified by the minimum inter-arrival time (also called period) T_i, the worst-case execution time C_i, and the relative deadline D_i [1], [16]. This indicates that a task τ_i in a task set τ infinitely invokes a series of jobs of τ_i, which satisfy as follows: the time to declare the completion of execution for any job of τ_i is at most C_i time units; the time instants at which two consecutive jobs of τ_i are invoked are separated at least T_i time units; and each job invoked by τ_i should be completed within D_i time units as a real-time constraint. We also assume that every τ_i has a constrained deadline, meaning $D_i \leq T_i$. A task set τ is scheduled by a given global, preemptive, work-conserving scheduling algorithm[1] on a platform of identical $m \geq 2$ processors. We consider FP scheduling under which a priority is assigned to each task rather than each job, meaning that all jobs invoked by a higher-priority task have equally higher priorities than those invoked by a lower-priority task. Let $LP(\tau_k)$ and $HP(\tau_k)$ denote a set of tasks each of whose priority is lower and higher than τ_k, respectively. Without loss of generality, we assume that a smaller task index i indicates a higher priority in FP scheduling; we also assume that one time unit indicates a single quantum.

The q-th job of τ_i (denoted by J_i^q) is released at r_i^q and finished at f_i^q; we use the notation J_i if it indicates an arbitrary job of a task τ_i. J_i^q has an absolute deadline $d_i^q = r_i^q + D_i$, meaning that J_i^q should complete its execution before or at d_i^q as a timing constraint. J_i^q is said to be schedulable if J_i^q can finish its execution before or at d_i^q; τ_i is said to be schedulable if every job J_i^q invoked by τ_i is schedulable; and a task set τ is said to be schedulable if every task $\tau_i \in \tau$ is schedulable. The response time R_i of a task τ_i is defined as $max_{\forall J_i^q}(f_i^q - r_i^q)$, and J_i^* is the job whose $(f_i^q - r_i^q)$ is the largest among all jobs J_i^q invoked by τ_i. Thus, a task set τ is schedulable, by the definition of the response time, if $R_i \leq D_i$ holds.

We assume that there are g different kinds of resources $\lambda = \{\lambda_1, \cdots, \lambda_g\}$ shared by different tasks. A lower-priority task holding a resource λ_x ($1 \leq x \leq g$) can be preempted by a higher-priority task, but a lower-priority task still holds λ_x until it explicitly releases λ_x. We consider *non-nested* shared resources, meaning that a job does not request for a shared resource while holding another one. Also, a resource λ_x cannot be used simultaneously by multiple tasks at a time instant. A task requests for multiple resources multiple times during its execution. $N_{i,x}$ denotes the maximum number of requests made by a single job J_i of τ_i for λ_x. For each request made by a job of τ_i for λ_x, let $C_{i,x}$ denote the maximum (or worst-case) duration for a job of τ_i to use resource λ_x by a single request; also, let $C_{i,x}^{sum}$ denotes the maximum (or worst-case) cumulative duration for a job of τ_i to use resource λ_x by all requests. $\lambda(\tau_i)$ represents the set of all resources used by jobs of τ_i. FT-PIP effectively promotes the priority of a job J_i^q during the job's execution to prevent the priority-inversion problem. Such a temporally-promoted priority is referred to as the *effective* priority compared to the *base* priority initially assigned by the FP scheduler in design time. $\lceil \lambda_x \rceil$ denotes the index of the highest base-priority task among all tasks that use λ_x. C_{ft}^x represents the time required to initiate the status of resource λ_x. Table 1 presents notations and their descriptions used throughout this paper.

We also assume that it is possible for some tasks to execute without any shared resource. For a resource locking protocol, we consider PIP [10], [17] for FP scheduling. Section III-B will detail how PIP operates.

[1] A scheduling algorithm is said to be global, preemptive and work-conserving, if the followings holds: each job can migrate from one core to another (*global*), each job can be preempted by a higher-priority jobs at any time (*preemptive*), and each processor is always kept busy as long as there is an unfinished, ready job (*work-conserving*).

TABLE 1. Notations and their descriptions.

Notation	Description	Notation	Description
n	the number of tasks in τ	λ	a set of resources
m	the number of processors	λ_x	a resource used by a task
τ	a task set	$N_{i,x}$	the maximum number of requests made by a single job J_i of τ_i for λ_x
τ_i	a task in τ	$C_{i,x}$	the maximum (worst-case) resource usage time among all requests for λ_x by jobs of τ_i
T_i	a minimum inter-arrival time or period of τ_i	$C_{i,x}^{sum}$	the summation of all resource usage time for λ_x by any single job J_i of τ_i.
C_i	the WCET of τ_i	$\lambda(\tau_i)$	the set of all resource types used by jobs of τ_i
D_i	a relative deadline of τ_i	$\lceil \lambda_x \rceil$	the index of highest base-priority task among all tasks that use λ_x
J_i^j	the j-th job invoked by τ_i	R_k^{ub}	an upper-bounded response time of τ_k
r_i^j	a release time of J_i^j	$LP(\tau_k)$	a set of tasks whose priorities are lower than τ_k
f_i^j	a finishing time of J_i^j	$HP(\tau_k)$	a set of tasks whose priorities are higher than τ_k
d_i^j	an absolute deadline of J_i^j	$N_i(\ell)$	the upper-bounded number of jobs fully executing for C_i in an interval of length ℓ
R_i	a response time of τ_i	$W_i(\ell)$	the upper-bounded workload in an interval of length ℓ
$I_k(a,b)$	the interference on τ_k in an interval [a, b)	$I_k^i(a,b)$	the interference of τ_i on τ_k in an interval [a, b)

III. FT-PIP

In this section, we introduce the flush task mechanism to prevent information leakage between two tasks associated with *noleak* relation by initiating the status of resources shared by the two tasks. Then, we propose FT-PIP that effectively incorporates the flush task mechanism into PIP.

A. FLUSH TASK WITH NOLEAK MODEL

For the security model, we adopt the *noleak* model designed for real-time systems in the previous study [5], which was derived from actual examples such as an aircraft system designed by the DO-178B standard [18] and the "RePLACE" system of Northrup Grumman [19]–[21].

To illustrate the underlying idea of the security model, we take an antenna controller software (ACSW) [22], [23] embedded in reconnaissance aerial vehicles as an example, which aims at obtaining the signal image of the target terrain; it transmits and receives radio frequency by controlling the reconnaissance antenna even when an optical image cannot be obtained in cloudy whether. ACSW mainly consists of five real-time tasks that are scheduled on a space-specific RTOS (Real-Time Operating System) such as real-time executive for multiprocessor systems (RTEMS) [24]. The network task (denoted by **tNet**) periodically receives a macro command (MCMD) from a ground station via the MIL-STD-1553B protocol [25] and inserts it into a MCMD queue after an integrity test. Then, the planning task (denoted by **tPlan**) retrieves each MCMD from the MCMD queue in a periodic manner and delivers proper information to other tasks corresponding to the MCMD. The mode task (denoted by **tMode**) conducts multiple operations regarding the internal-mode transition (e.g., high-resolution or wide-range mode) of the reconnaissance aerial vehicle, which is performed according to the MCMD. The utility task (denoted by **tUtil**) executes multiple work such as FDIR (fault detection, isolation and recovery) and formats network packets containing information of operation results and the current status of the system that will be transmitted to the ground station. The preparation task (denoted by **tPre**) is performed for the operation preparation whenever the other tasks are not performed. Table 2 presents task parameters of ACSW: T_i and D_i are determined by a system designer, and WCET (i.e., C_i) is measured in various actual operation scenarios on the target system with 256MB SDRAM and a multi-processor platform based on FT Leon3 CPU architecture (80Mhz clock speed). As shown in Table 2, periods of **tPlan**, **tNet**, **tMode** and **tUtil** are harmonic,[2] and that of **tPlan** is ∞ because it executes whenever no other tasks are active. The relative deadline and WCET of each task are smaller than its period. The characteristic of **tFT(cache)** will be explained later.

TABLE 2. The task parameters of ACSW in milliseconds.

	T_i	D_i	WCET
tPlan	62.5	50	2.98
tNet	125	100	0.54
tMode	250	200	30.08
tUtil	500	400	231.72
tPre	∞	∞	∞
tFT(cache)	∞	∞	0.5

The *noleak* model is inspired by the fact that the embedded software such as ACSW is normally composed of multiple subsystems developed by different vendors, and the information leakage from a security-sensitive task of one vendor to another task of a different vendor should be prohibited. For example, **tNet** could have a higher security level since it receives a MCMD containing sensitive mission-relevant information from a ground station, and it is undesirable for the other tasks such as **tPlan** to gain the unintended

[2]Periods are called harmonic if one is a divisor or multiple of another

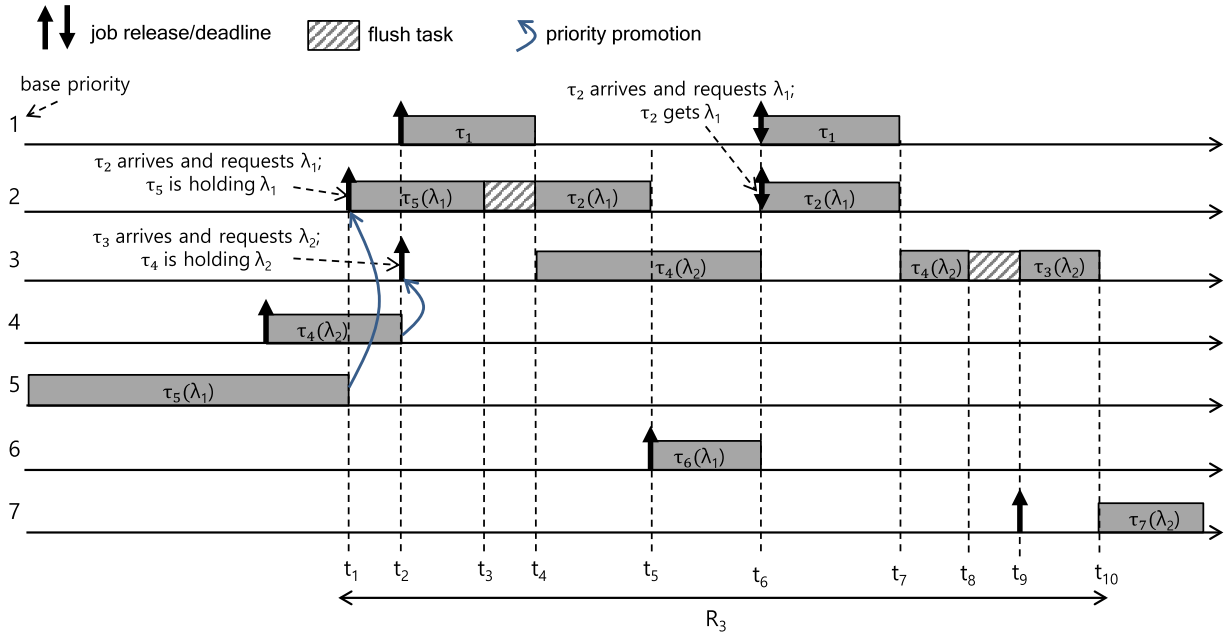


FIGURE 1. An example schedule under the FT incorporated priority-inheritance protocol for $m = 2$.

TABLE 3. An example of noleak relations associated with ACSW.

from \ to	tPlan	tNet	tMode	tUtil	tPre
tPlan	-	T	T	T	T
tNet	T	-	T	T	T
tMode	F	F	-	F	F
tUtil	F	F	F	-	F
tPre	F	F	F	F	-

information. Such a constraint for a resource λ_x is presented by the *noleak* relation that is denoted by *noleak*(**tNet**, **tPlan**, λ_x). Table 3 shows *noleak* relations associated with the five tasks of ACSW. "T" in Table 3 means that information leakage from the corresponding task in the same row to the task in the same column should be prevented, while "F" means the opposite case.

The main idea of FT is to conditionally initialize the resources shared by the multiple tasks, which can satisfy the security constraints. The FT mechanism incurs a timing penalty for such initialization, and the operation and duration for each FT are determined by which shared resource is utilized for the considered situation. To illustrate the above example, we consider shared cache (32KB for data and instruction caches, respectively) of FT Leon3 CPU. As described, **tNet** and **tPlan** inserts and retrieves MCMDs in the MCMD queue in every 125 and 62.5 milliseconds, respectively, and thus related instructions and data are periodically stored in the shared cache. That is, the location of the MCMD queue is assessed periodically, and it can be inferred by the carefully designed cache-based side-channel attack illustrated in Section II-A. Therefore, such information that remains in the shared cache is needed to be initiated by the cache-related FT (denoted by **tFT(cache)**) after a job of **tNet** or **tPlan** completes its execution; note that Table 3 presents such constraints. The cache-related FT overwrites cache sets regarding the information of the MCMD queue with zero data, or it just supplies null voltage to the cache for a short duration. Table 2 also includes the task parameters of **tFT(cache)** whose WCET is 0.5 milliseconds. **tFT(cache)** executes non-preemptively until it completes its execution without a notion of deadline.

B. PROTOCOL

PIP effectively promotes the priority of a job J_k of a task τ_k holding a shared resource λ_x so that J_k can finish its execution without yielding λ_x before its completion. The mechanism of PIP guarantees mutual execution for a critical section that exists during usage of a share resource λ_x.

The priority-inheritance under PIP occurs when J_k is holding a shared resource λ_x for its execution, and another higher-priority job J_i of a task τ_i requests λ_x; in this case, the priority of J_k is promoted to i. We assume that the effective priority of a task is initially equal to the base priority of the task, and a temporally-promoted effective priority of a task is set back to the base priority of the task at the next job release time of the task.

Hence, the execution of J_i can be also hindered by other lower-priority jobs under PIP because the effective priority (promoted from the base priority by the priority-inheritance policy) of a lower-priority job can be higher than τ_i.

Figure 1 presents an example schedule of $\tau = \{\tau_1, \cdots, \tau_5\}$ with their base-priorities represented by task indexes 1–5 under PIP for $m = 2$. The example assumes that τ_2, τ_5 and τ_6 share λ_1. Also, τ_3, τ_4 and τ_7 share λ_2, and τ_1 executes without any shared resource. The table of *noleak* relations is constructed for each resource, λ_1 and λ_2, respectively;

$noleak(\tau_5, \tau_2, \lambda_1) = T$ and $noleak(\tau_4, \tau_3, \lambda_2) = T$ are hold, and the other relations are all F. Note that each task may request multiple types of shared resources multiple times in our system model, but each task in the example requests a single shared resource during its execution for simplicity.

At the beginning, τ_5 and τ_4 perform their executions with λ_1 and λ_2. Then, τ_2 arrives and requests λ_1 at t_1, at which the effective priority of τ_5 is promoted to 2 because τ_5 is holding λ_1. At t_2, τ_1 and τ_3 arrive, and τ_3 requests λ_2, which induces that the effective priority of τ_4 is promoted to 3; this is because τ_4 is holding λ_2, and τ_4 is preempted by τ_1 due to its effective priority lower than that of both τ_1 and τ_5. At t_3, τ_5 completes its execution, and a FT is invoked to initiate the status of λ_1 before τ_2 begins its execution by holding λ_1. At t_4, τ_1 and a FT finish their executions, and τ_2 with λ_1 and τ_4 with λ_2 start their executions. Then, their executions are completed at t_5 and t_6, respectively. Also, τ_6 starts its execution with λ_1 at t_5, and a FT is not invoked according to the predefined *noleak* relations. At t_6, τ_1 and τ_2 with λ_1 start their executions and complete them at t_7. Thereafter, τ_4 with λ_2, a FT and τ_3 with λ_2 begin their execution at t_7, t_8 and t_9, respectively. At t_9, τ_7 arrives and requests λ_2, but its execution begins after τ_3 finishes its execution at t_{10}.

IV. RTA FRAMEWORK FOR FT-PIP

In this section, we first review the existing RTA framework for FP scheduling on a multiprocessor platform that does not consider shared resources and FTs. We then propose a new RTA framework for the FP scheduling under FT-PIP to judge schedulability of a set of real-time tasks.

A. EXISTING RTA FRAMEWORK

The underlying principle of the RTA framework is to find an upper bound of the interval length between the release time and the finishing time of every job of a task τ_k. Suppose that we focus on a job of τ_k, called J_k^q, whose release time is r_k^q. For J_k^q not to execute at a time slot in an interval of $[r_k^q, r_k^q+\ell)$, m other jobs should be executed. To express such interference, let $I_k^i(r_k^q, r_k^q + \ell)$ denote the amount of interference of τ_i on τ_k in an interval $[r_k^q, r_k^q + \ell)$, which is defined as the length of cumulative intervals in $[r_k^q, r_k^q + \ell)$ such that J_k^q cannot execute but jobs of τ_i execute. Considering that m other higher-priority jobs are needed for J_k^q not to be executed at a time slot, the length of cumulative intervals in $[r_k^q, r_k^q+\ell)$ such that J_k^q cannot execute owing to the executions of other jobs can be represented by $\left(\sum_{\tau_i\in\tau\setminus\{\tau_k\}} I_k^i(r_k^q, r_k^q + \ell)\right)/m$. Also, out of ℓ time slots in $[r_k^q, r_k^q+\ell)$, we focus on the $(\ell-C_k+1)$ time slots. If J_k^q cannot be executed in the $(\ell - C_k + 1)$ time slots, it can execute at most for $(C_k - 1)$ time units, which means J_k^q cannot finish its execution until $r_k^q + \ell$. Since we focus on the $(\ell - C_k + 1)$ time slots, the amount of execution of jobs of τ_i at those slots is upper-bounded by $(\ell - C_k + 1)$. This reduces the amount of calculated interference from $\left(\sum_{\tau_i\in\tau\setminus\{\tau_k\}} I_k^i(r_k^q, r_k^q+\ell)\right)/m$ to $\left(\sum_{\tau_i\in\tau\setminus\{\tau_k\}} min\left(I_k^i(r_k^q, r_k^q+\ell), \ell - C_k + 1\right)\right)/m$ [3], [26]. Therefore, if the sum of the WCET of J_k^q itself and its interference at the $(\ell - C_k + 1)$ time slots in $[r_k^q, r_k^q + \ell)$ is no larger than ℓ, J_k^q finishes its execution before $r_k^q + \ell$, which is recorded in the following lemma.

Lemma 1 (From [3], [26]): A task $\tau_k \in \tau$ is schedulable, if every job J_k^q satisfies the following inequality for at least one $C_k \le \ell \le D_k$.

$$C_k + \left\lfloor \frac{1}{m} \sum_{\tau_i\in\tau\setminus\{\tau_k\}} min\left(I_k^i(r_k^q, r_k^q + \ell), \ell - C_k + 1\right) \right\rfloor \le \ell. \tag{1}$$

Since the exact calculation of $I_k^i(r_k^q, r_k^q + \ell)$ is impossible before run-time, existing studies developed an upper bound of $I_k^i(r_k^q, r_k^q + \ell)$. The upper bound is a function of the task parameters of τ_i, those of τ_k and the interval length ℓ only, and it should be calculated without requiring any run-time information. Then, the RTA framework finds ℓ that satisfies Eq. (1) as follows. At the beginning, ℓ is set to C_k, and the task is deemed schedulable if the inequality (i.e., Eq. (1)) holds. Otherwise, ℓ is set to the previous value of the LHS of the inequality, and it tests the inequality again; it repeats the procedure until the inequality holds or ℓ becomes larger than D_k. During such iterations, it judges that τ_k is deemed schedulable if it finds a value of ℓ in $[C_k, D_k]$ that satisfies the inequality, and deemed unschedulable if ℓ becomes larger than D_k. Then, the value of ℓ in the former (if any) is denoted by R_k^{ub}, which is an upper-bound of R_k (i.e., the response time of τ_k).

B. NEW RTA FRAMEWORK FOR FT-PIP

In this subsection, we propose a new RTA framework for FT-PIP by extending the existing RTA framework introduced in the previous subsection. To this end, we need to investigate how the target resources shared by multiple tasks and the FT mechanism that conditionally initiates the status of shared resources affect the value of R_k^{ub} under our system model described in Section II. The upper-bounded response time R_k^{ub} under FT-PIP is determined by the following five factors:

(i) the worst-case execution C_k,
(ii) the total amount of time units in which J_k^q waits to get all resources $\lambda_x \in \lambda(\tau_k)$,
(iii) the total amount of time units in which J_k^q's execution is interfered because of FTs to initiate the status of all resources $\lambda_x \in \lambda(\tau_k)$,
(iv) the total amount of time units in which J_k^q's execution is hindered due to either other jobs that request their required resources $\lambda_y \in \lambda\setminus\lambda(\tau_k)$ or other jobs that do not require any resource, whose effective or base priorities are higher than J_k^q's one, and
(v) the total amount of time units in which J_k^q's execution is interfered due to FTs to initiate the status of $\lambda_y \in \lambda \setminus \lambda(\tau_k)$.

While (i) is given by our system model, (ii) is upper-bounded by the following two terms:

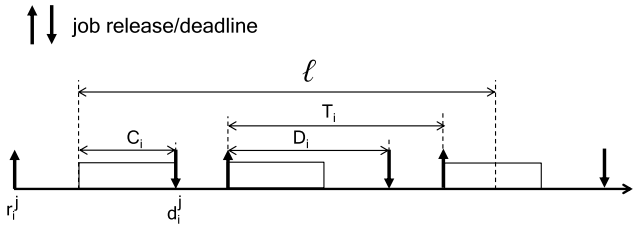


FIGURE 2. The worst-case scenario in which the amount of execution of jobs of τ_i performed in an interval of length ℓ is maximized.

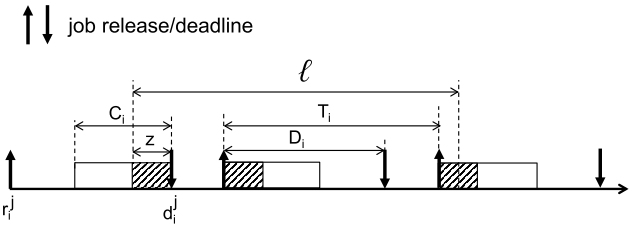


FIGURE 3. The worst-case scenario in which the amount of execution of jobs of τ_i for the portion z performed in an interval of length ℓ is maximized.

- IL_k: the upper-bounded total amount of time that a job J_k^q waits to get its required resource λ_x while λ_x is used by lower base-priority (whose effective priority may be higher than or equal to k) jobs, and
- $IH_k(\ell)$: the upper-bounded total amount of time that J_k^q waits to get its required resource λ_x while λ_x is used by higher base-priority jobs in an interval of length ℓ.

Next, (iii) is upper-bounded by the following term:

- $FT_k(\ell)$: the upper-bounded total amount of time in which J_k^q's execution is interfered due to FTs to initiate the status of all resources $\lambda_x \in \lambda(\tau_k)$ in an interval of length ℓ.

Also, (iv) can be upper-bounded by the following three terms:

- $IL'_k(\ell)$: the upper-bounded total amount of time that lower base-priority jobs (but with higher effective-priority) J_i^q execute with their required resources $\lambda_y \in \lambda(\tau_i \in LP(\tau_k))$ in an interval of length ℓ,
- $IH'_k(\ell)$: the upper-bounded total amount of time that higher base-priority jobs J_i^q execute with their required resources $\lambda_y \in \lambda(\tau_i \in HP(\tau_k))$ in an interval of length ℓ, and
- $IH''_k(\ell)$: the upper-bounded total amount of time that higher base-priority jobs J_i^q execute without any resource in an interval of length ℓ.

Finally, (v) is upper-bounded by the following term:

- $FT'_k(\ell)$: the upper-bounded total amount of time units in which J_k^q's execution is interfered due to FTs to initiate the status of $\lambda_y \in \lambda \setminus \lambda(\tau_k)$ in an interval of length ℓ.

To upper-bound the execution performed in an interval of length ℓ, we use the notion of workload for each task τ_i proposed in [3]. As shown in Figure 2, in the scenario in which the amount of execution of τ_i is maximized in an interval of length ℓ, the first job begins its execution at the left-most of the interval of length ℓ, and finishes at its absolute deadline d_i^q; thereafter, the successive jobs start its execution as early as possible. The amount of execution of jobs of τ_i in an interval of length ℓ under the scenario is called workload $W_i(\ell)$. To express the upper-bound of execution during holding a resource $\lambda_x \in \lambda(\tau_i)$, we generalize the notion of workload as follows. As shown in Figure 3, the portion z of C_i of each task is maximally included in an interval of length ℓ such that the execution for z is performed as late as possible for the left-most job in the interval, and successive jobs begin their executions for z as early as possible.

Let $N_i(\ell, z)$ is the number of jobs whose execution for the portion z is fully performed in an interval of length ℓ, and $W_i(\ell, z)$ is cumulative execution performed for z included in jobs of a task τ_i. $N_i(\ell, z)$ and $W_i(\ell, z)$ can be upper-bounded as follows:

$$N_i(\ell, z) = \left\lfloor \frac{\ell - z + D_i}{T_i} \right\rfloor, \tag{2}$$

$$W_i(\ell, z) = z \cdot N_i(\ell, z) + min\big(z, \ell - z + D_i - T_i \cdot N_i(\ell, z)\big). \tag{3}$$

The execution of J_k^q with its required resources λ_x is hindered when λ_x is used by other jobs or the status of λ_x is initiated by a FT, even if there are multiple idle processors. On the other hand, m jobs (or FTs) are needed to hinder J_k^q's execution when such jobs (or FTs) do not use λ_x. Based on the reasoning, we derive the following theorem.

Theorem 1: A task τ_k scheduled by FP scheduling with FT-PIP can be schedulable, if every job J_k^q of τ_k satisfies the following inequality for at least one $C_k \le \ell \le D_k$.

$$C_k + IL_k + IH_k(\ell) + FT_k(\ell) + \left\lfloor \frac{IH'_k(\ell) + IL'_k(\ell) + IH''_k(\ell) + FT'_k(\ell)}{m} \right\rfloor \le \ell. \tag{4}$$

Then, how to find ℓ for the theorem is the same as the procedure explained in Section IV-A.

We now derive each term in Eq. (4) as follows; note that C_k is given from our system model. During execution of J_k, a resource $\lambda_x \in \lambda(\tau_k)$ is requested $N_{k,x}$ times. If multiple jobs of tasks τ_i in $LP(\tau_k)$ request λ_x during their execution, the waiting time to get a resource λ_x for J_k is upper-bounded by the maximum value among $C_{i,x}$ of tasks τ_i in $LP(\tau_k)$. Thus, IL_k is calculated as

$$IL_k = \sum_{\lambda_x \in \lambda(\tau_k)} N_{k,x} \cdot \max_{\left(\tau_i \in LP(\tau_k)\right) \wedge \left(\lambda_x \in \lambda(\tau_k) \cap \lambda(\tau_i)\right)} C_{i,x}. \tag{5}$$

While IL_k is determined by how many times J_k requests a resource $\lambda_x \in \lambda(\tau_k)$ during its execution, $IH_k(\ell)$ is dependent on how long a job J_i of a task $\tau_i \in HP(\tau_k)$ executes with $\lambda_x \in \lambda(\tau_k)$ in an interval of length ℓ. This is because usage of $\lambda_x \in \lambda(\tau_k)$ for J_k is guaranteed after the lower-priority job executing with λ_x releases λ_x, while it is not if a higher-priority job is holding λ_x and another higher-priority job is waiting to get λ_x. Using $C_{i,x}^{sum}$ that represents the maximum (or worst-case)

cumulative duration for a job of τ_i to use resource λ_x by all requests, we derive $IH_k(\ell)$ as follows:

$$IH_k(\ell) = \sum_{\tau_i \in HP(\tau_k)} W_i\Big(\ell, \sum_{\lambda_x \in \lambda(\tau_i) \cap \lambda(\tau_k)} C_{i,x}^{sum}\Big). \quad (6)$$

We will define function $maxflow(\tau_k, \ell, \lambda_x)$ that upper-bounds the number of FTs invoked in $[r_k, r_k+\ell)$ to initiate the status of a resource λ_x in the next section. Using the function, $FT_k(\ell)$ is calculated as

$$FT_k(\ell) = \sum_{\lambda_x \in \lambda(\tau_k)} maxflow(\tau_k, \ell, \lambda_x) \cdot C_{ft}^{x}. \quad (7)$$

Suppose that a job of a lower-priority task $\tau_i \in LP(\tau_k)$ uses a resource (that is not used by τ_k) $\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)$, and there is at least one task τ_j that uses λ_y and has a priority higher than τ_k (i.e., $\tau_j \in HP(\tau_k)$). Then, τ_i's priority can be promoted to a priority higher than τ_k due to the priority-inheritance policy of FT-PIP, thereby contributing to the interference on τ_k. Thus, $IL'_k(\ell)$ is calculated by

$$IL'_k(\ell) = \sum_{\tau_i \in LP(\tau_k)} W_i\Big(\ell, \sum_{\left(\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)\right) \wedge \left(\lceil \lambda_y \rceil < k\right)} C_{i,y}^{sum}\Big). \quad (8)$$

The execution of τ_k can be hindered in a time slot when there are m jobs of higher-priority tasks $\tau_i \in HP(\tau_k)$ that use $\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)$ in the time slot. Hence, the following derive $IH'_k(\ell)$ and $IH''_k(\ell)$.

$$IH'_k(\ell) = \sum_{\tau_i \in HP(\tau_k)} W_i\Big(\ell, \sum_{\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)} C_{i,y}^{sum}\Big) \quad (9)$$

$$IH''_k(\ell) = \sum_{\tau_i \in HP(\tau_k)} W_i\Big(\ell, C_i - \sum_{\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)} C_{i,y}^{sum}\Big) \quad (10)$$

Using the function $maxflow(\tau_k, \ell, \lambda_y)$, we calculate FT'_k as follows.

$$FT'_k(\ell) = \sum_{\tau_i \in \tau \setminus \tau_k} \sum_{\lambda_y \in \lambda(\tau_i) \setminus \lambda(\tau_k)} maxflow(\tau_k, \ell, \lambda_y) \cdot C_{ft}^{y} \quad (11)$$

V. FT BOUNDS

In the previous section, we upper-bound the amount of executions of higher effective- or base-priority jobs performed in an interval of length ℓ to develop a new RTA for FT-PIP. The remaining issue is to derive the $maxflow(\tau_k, \ell, \lambda_x)$ function for calculating $FT_k(\ell)$ and $FT'_k(\ell)$. In this section, we transform the problem of bounding the number of FTs to initiate the status of λ_x and λ_y invoked during the execution of a job J_k of a task τ_k under FT-PIP, to the max-flow problem to calculate $FT_k(\ell)$ and $FT'_k(\ell)$. Focusing on the previous study [6] initially proposed for non-preemptive scheduling that considers a single resource without mutual exclusion on uniprocessor platforms, we extend its approach to FT-PIP that considers multiple types of resources with mutual exclusion on multiprocessor platforms.

We have the following observation regarding FT invocation under PIP.

Observation 1: A FT for λ_x is invoked, only when a job J_i completes the usage for λ_x, a successive job J_j begins the usage for λ_x, and there is a *noleak* relation $noleak(\tau_i, \tau_j, \lambda_x) = T$.

Between the start and completion of execution of J_i^q that holds a resource $\lambda_x \in \lambda(\tau_i)$, only two types of schedule events can be made: preemption and priority-inheritance. J_i^q can be preempted by a higher effective- or base-priority job, but it does not yield λ_x until J_i^q completes its execution (e.g., between t_2 and t_4 in Figure 1). Also, J_i^q's effective priority can be promoted by the policy of PIP when another job J_j^p has a higher priority and requests the same resource $\lambda_x \in \lambda(\tau_i)$, but it continues its execution by holding λ_x (e.g., t_1 in Figure 1). Thus, a FT is invoked to initiate the status of λ_x only when a job J_i^q completes its usage for λ_x and another job J_j^q begins its usage for λ_x in accordance with its *noleak* relation, i.e., $noleak(\tau_i, \tau_j, \lambda_x) = T$.

Observation 1 implies that the number of FTs invoked to initiate the status of $\lambda_x \in \lambda(\tau_k)$ in an interval of length ℓ depends on (i) the number of resource requests for λ_x of each job, (ii) execution sequence and (iii) *noleak* relations of them. For example, Figure 4 illustrates the execution patterns of four tasks from τ_1 to τ_4 with a shared resource in an interval of length ℓ. As shown in Figure 4, each job of τ_1, τ_2, τ_3, and τ_4, requests λ_1 one, two, one, and three times, respectively. Then, a FT can be invoked when a job starts its execution with λ_1. Thus, the number of invoked FTs for λ_1 in the interval of length ℓ in Figure 4 cannot be greater than eight because λ_1 is used nine times if we restrict the scope of interest to the interval of length ℓ.

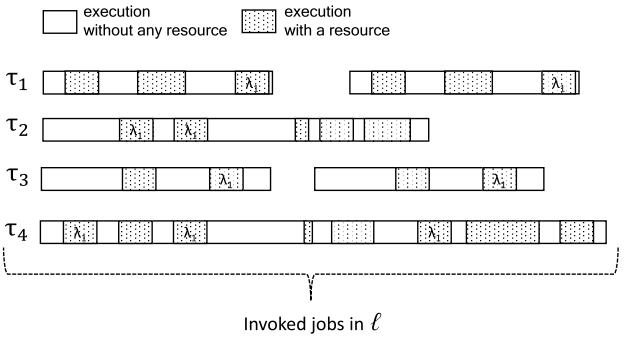


FIGURE 4. An example of invoked jobs and their resource request patterns in an interval of length ℓ.

To simplify the problem, we address (ii) by finding a resource-usage sequence producing the maximum number of FTs invoked among possible permutations of the sequences in an interval of length ℓ. To this end, we transform our problem to the max-flow problem by utilizing a notion of *noleak* graph.

The followings are the construction rules of the *noleak* graph for λ_x in the interval of length ℓ.

- For a task τ_i and the number of requests for λ_x in the interval of length ℓ (denoted by $N_{i,x}(\ell)$), sending and receiving nodes (denoted by $send_i^{\alpha}$ and $recv_i^{\alpha}$,

TABLE 4. An example of noleak relations for the example in Figure 4.

from \ to	τ_1	τ_2	τ_3	τ_4
τ_1	-	F	T	F
τ_2	T	-	F	T
τ_3	T	F	-	F
τ_4	F	F	T	-

for $1 \leq \alpha \leq N_{i,x}(\ell)$, respectively) are created. $N_{i,x}(\ell)$ is obtained by multiplying $N_{i,x}$ and the number of τ_i's jobs in ℓ.

- A directed edge from $send_i^{\alpha}$ to $recv_j^{\beta}$ is created if $noleak(\tau_i, \tau_j, \lambda_x) = T$, for $1 \leq \beta \leq N_{i,x}(\ell)$.

The followings are the constraints of directed and FT edges of a *noleak* graph.

- A sending node $send_i^{\alpha}$ has at most one FT edge.
- A receiving node $recv_j^{\beta}$ has at most one FT edge.

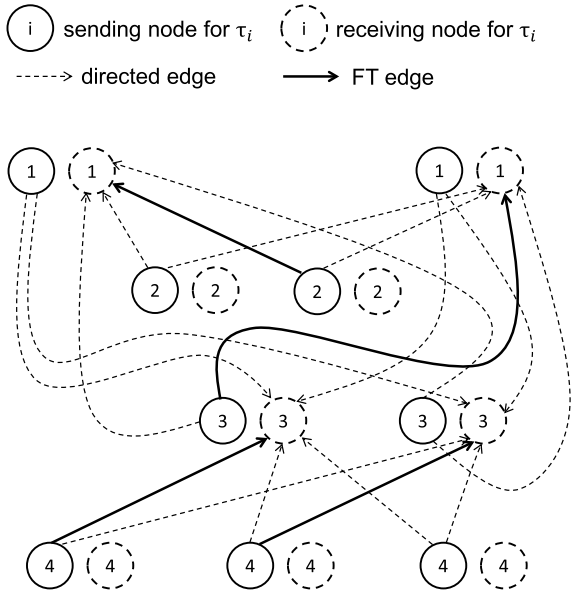


FIGURE 5. An example of the noleak graph for the example in Figure 4.

Then, $maxflow(\tau_k, \ell, \lambda_x)$ upper-bounds the number of invoked FTs for λ_x in an interval of length ℓ (e.g., $[r_k^*, r_k^* + \ell)$) as described in Figure 5, which illustrates an example of the *noleak* graph generated for λ_x shared by four tasks in an interval of length ℓ in Figure 4. In Figure 5, a circle drawn with a solid line represents a sending node, and that with a dotted line represents a receiving node. Also, the number in a circle represents the task index of the corresponding job. A directed dotted line from a sending node $send_i^{\alpha}$ to a receiving node $recv_j^{\beta}$ represents a directed edge, which indicates that there is a *noleak* relation $noleak(\tau_i, \tau_j, \lambda_x) = T$. A directed solid line from a sending node $send_i^{\alpha}$ to a receiving node $recv_j^{\beta}$ represents a FT edge, which implies that a FT is invoked between J_i and J_j.

VI. DISCUSSION

For an identical multi-processor platform, constrained-deadline tasks (i.e., $D_i \leq T_i$), and a real-time constraint, it is well-known that the problem of finding a feasible (i.e., schedulable) schedule (resulted from any scheduling algorithm) is NP-hard in a strong sense [34]. As we consider timing, security constraints, and mutual exclusion simultaneously, it should be NP-hard when it comes to finding a scheduling algorithm that always yields a feasible schedule if it exists. Thus, we consider a heuristic approach such as the fixed-priority scheduling as a number of existing studies did. That is, FT-PIP is heuristic for the problem of real-time scheduling algorithm that is NP-hard. When it comes to the problem of schedulability analysis, to find the exact response time is NP-hard in a strong sense as well even for the uni-processor case [35], and thus it is naturally NP-hard for our multi-processor case. To address the problem, we adopt RTA whose time complexity is $O(n^2 \cdot maxT)$ (which is pseudo polynomial) where n is the number of tasks in a task set τ and $maxT$ is the maximum among all tasks' periods in τ. Then, the max-flow problem to upper-bound the number of invoked flush tasks in each iteration of RTA is solved in $O(n^3)$ (e.g., Ford-Fulkerson algorithm), thereby resulting in the time complexity $O(n^5 \cdot maxT)$ for our proposed schedulability analysis.

There are a number of other defense methods such as schedule randomization protocol for real-time systems. For example, TaskShuffler [29] is a well-developed schedule randomization protocol, and it would be very interesting if we can compare the performance of TaskShuffler and our proposed method with the same performance metric. The underlying idea of TaskShuffler is to randomize the scheduling pattern of the given scheduling algorithm without any deadline miss, and the degree of randomness is measured by a notion of schedule entropy. To express the schedule entropy of our proposed method and justify it would be potential following work of our study.

VII. EVALUATION

In this section, we evaluate the effectiveness of FT-PIP under varying system parameters in our experiment environment. To this end, we randomly generate a number of task sets according to the well-known task-set generation method referred to as UUnifast-discard [27]. To generate a task set, UUnifast-discard basically has three input parameters: (i) the number of processors m (4 and 8), (ii) the number of tasks n ($m + 1$, $1.5m$, $2m$, $2.5m$, $3m$, $3.5m$, and $4m$) in each task set, and (iii) the task set utilization U ($0.1m$, $0.2m$, $0.3m$, $0.4m$, $0.5m$, and $0.6m$) that is defined as $\sum_{\tau_i \in \tau} C_i/T_i$. For a given task utilization for τ_i (denoted by u_i) assigned by UUnifast-discard with given choices for (i), (ii) and (iii), T_i is randomly selected in [1, 1000], C_i is computed as $u_i \cdot T_i$, and D_i is set to T_i. We additionally consider an input parameter for our resource model: (iv) the number of considered resources γ ($1/4m$, $1/2m$, and m).

Then, $N_{i,x}$, $C_{i,x}^{sum}$, and $C_{i,x}^{max}$ are determined as follows. $N_{i,x}$ is uniformly selected in $\{1, 2, 3\}$; $C_{i,x}^{sum}$ is determined by a random real value in $[0.1, 1] \cdot C_i/\gamma$; and $C_{i,x}^{max}$ is set to $C_{i,x}^{sum}/N_{i,x}$. Also, we consider (v) the flush cost C_{ft}^x $(2, 3, 4, \cdots, 19, 20)$, defined as the time required to initiate the status of the shared resource. For a given value C_{ft}^x, we assume that every considered resource in a task set requires the same C_{ft}^x to cleanse its information. Next, each element of the n-by-n *noleak* table for each resource $\lambda_x \in \lambda$ is randomly set to either 0 or 1. Thereafter, the element intersected by the two tasks τ_i and τ_j in the *noleak* table for λ_x is set to 0 if they do not share the resource λ_x. For each combination of input parameters (i)–(v), we generate 1,000 task sets, thereby resulting in $2 \cdot 7 \cdot 6 \cdot 3 \cdot 19 = 4,788,000$ task sets in total. As a performance metric, we measure the number of task sets deemed schedulable by the proposed schedulability test in Theorem 1.

To investigate the performance of our proposed framework in varying parameter settings, we consider the following three RTA tests for the RM (Rate-Monotonic) scheduling algorithm, under which a task with a smaller period has a higher priority than a task with a larger period.

- **PIP**: RTA in [15] for PIP incorporated RM without considering a security constraint.
- **FT-PIP-ob**: RTA in Theorem 1 for FP-PIP incorporated RM considering a security constraint, using a naive upper-bound for $FT_k(\ell)$ and $FT'_k(\ell)$, which is the number of jobs executed before J_k of τ_k completes its execution in an interval of length ℓ, i.e., $\sum_{\tau_i \in HP(\tau_k)} \lceil \frac{\ell}{T_i} \rceil$.
- **FT-PIP-mf**: RTA in Theorem 1 for FT-PIP incorporated RM considering a security constraint, using Eqs. (7) and (11) for $FT_k(\ell)$ and $FT'_k(\ell)$, respectively.

Figures 6(a)-(h) show the experiment results for $m = 4$ with various input parameter settings. In particular, the figure plots the number of tasks deemed schedulable by the corresponding schedulability tests over varying task set utilization U (in Figures 6(a) and (b)), different number of tasks n (in Figures 6(c) and (d)), varying number of resources γ (in Figures 6(e) and (f)), and different flush cost C_{ft}^x (in Figures 6(g) and (h)). The caption of each figure represents the input parameter settings other than the considered one shown in the x-axis of each figure. For example, the input parameter settings for Figure 6(a) are $m = 4$, $n = 6$, $\gamma = 1$ and $C_{ft}^x = 2$, and the figure's x-axis represents varying U (i.e., the task set utilization). Note that, as mentioned earlier, we generate 1,000 task sets for each input parameter setting, and thus each point in each figure in Figure 6 represents the number of tasks deemed schedulable among 1,000 generated tasks; it does not indicate the average value of multiple tries.

We have the following observations from Figure 6.

*O*1. There is a larger performance gap between **PIP** and **FT-PIP-ob** while **FT-PIP-mf** moderately narrows down the gap under every input parameter setting, as shown in all figures in Figure 6.

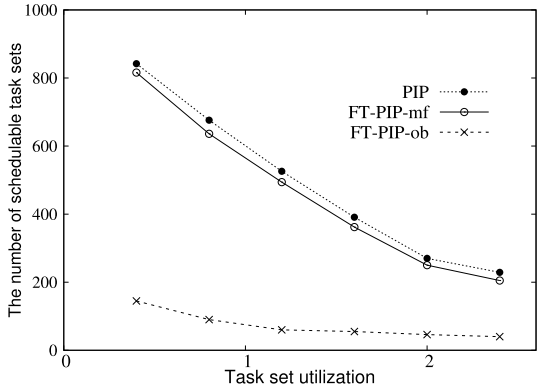
*O*2. The performance of **FT-PIP-mf** and **FT-PIP-ob** decreases by increasing values of U, n, γ, and C_{ft}^x, as shown in all figures in Figure 6.

*O*3. The performance of **PIP** is not affected by the value of C_{ft}^x (as shown Figures 6(a)-(f)), but it is affected by the value of γ (as shown Figures 6(g)-(h)).

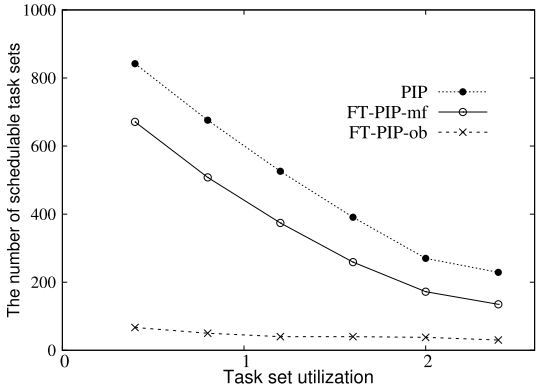
*O*1 demonstrates **FT-PIP-mf**'s superior performance in upper-bounding the number of FTs invoked during a job J_k^q's execution by translating it to the max-flow problem. On the other hand, **FT-PIP-ob** derives it by counting the number of jobs executed before J_k^q completes its execution, which worsens the performance of **PIP** significantly. For example, for $U = 0.4$ in Figure 6(a), the number of schedulable task sets under **PIP**, **FT-PIP-mf** and **FT-PIP-ob** are 842, 816, and 145, respectively, out of 1000; thus, **FT-PIP-ob** obtains only $145/842 = 17.2\%$ of the performance of **PIP** while **FT-PIP-mf** does $816/842 = 96.9\%$.

Increasing values of U, n, γ, and C_{ft}^x result in *O*2 with their own different reasons. As the value of U (with the fixed value of n) increases, the utilization (C_i/T_i) of each task in a task set may increase, and this increases the worst-case execution time C_i and the worst-case interference from higher-priority tasks for each task. Thus, the worst-case response time of each job tends to increase with a high probability. The larger value of n (with the fixed value of U) also produces a smaller number of task sets deemed schedulable under the corresponding schedulability tests. This is because the corresponding schedulability tests are based on the notion of workload $W_i(\ell)$ and $W_i(\ell, z)$ in an interval of length ℓ (as illustrated in Figures 2 and 3). The notion of workload is used to derive the worst-case interference from higher-priority tasks, and it calculates the upper-bounded (not exact) amount of execution of a higher-priority task in an interval of length ℓ. Thus, as the number of tasks in a task set increases, the pessimistic calculation of the interference stands out more. A larger value of γ results in the decreased performance of the corresponding schedulability tests owing to a similar reason. **FT-PIP-mf** and **FT-PIP-ob** derive the worst-case waiting time to obtain a certain shared resource even though the worst-case scenario may not always happen. Also, a *noleak* table is required for each resource, implying that the number of FTs invoked during the execution of a job may increase. These two factors inevitably decrease the performance of the corresponding schedulability analysis. Finally, the increased value of C_{ft}^x degrades the possibility of timely execution because it surely increases the worst-case response time of each job when the upper-bounded number of FTs is not changed.

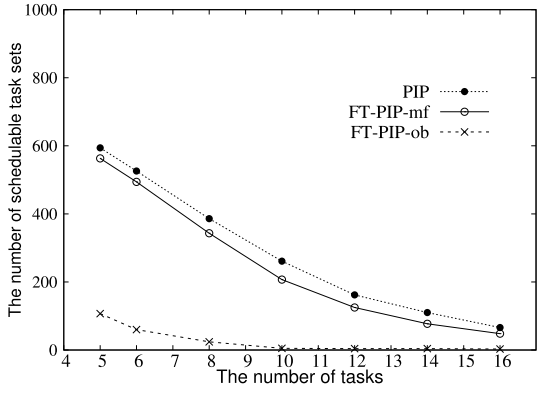
*O*3 holds due to the property of **PIP**. That is, **PIP** does not consider the security constraint, and thus its performance is not affected by the value of C_{ft}^x as shown in Figures 6(g) and (h). On the other hand, it uses the same techniques as those used by **FT-PIP-mf** and **FT-PIP-ob** to derive the worst-case response time for each job. Therefore its performance decreases with increasing value of U, n, and γ due to the same reasons explained so far.



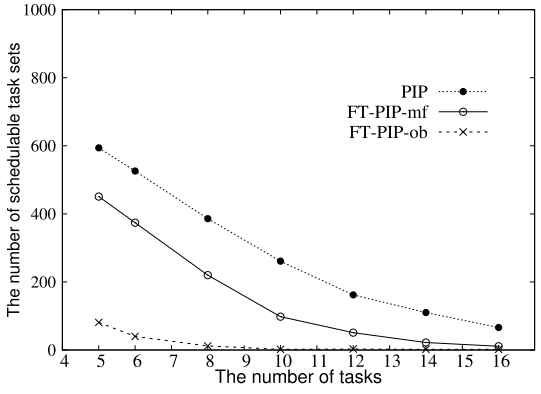
(a) $m = 4, n = 6, \gamma = 1, C_{ft}^{x} = 2$



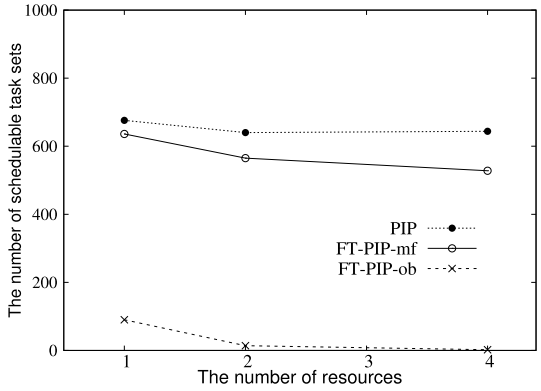
(b) $m = 4, n = 6, \gamma = 1, C_{ft}^{x} = 10$



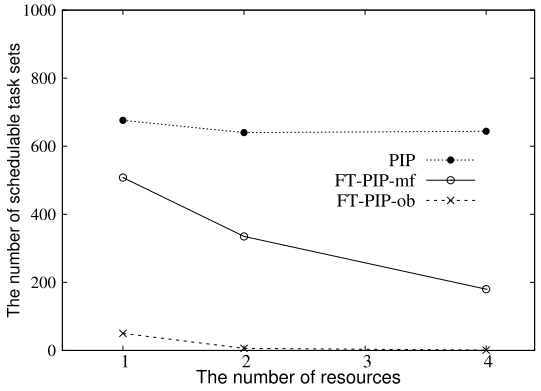
(c) $m = 4, U = 1.2, \gamma = 1, C_{ft}^{x} = 2$



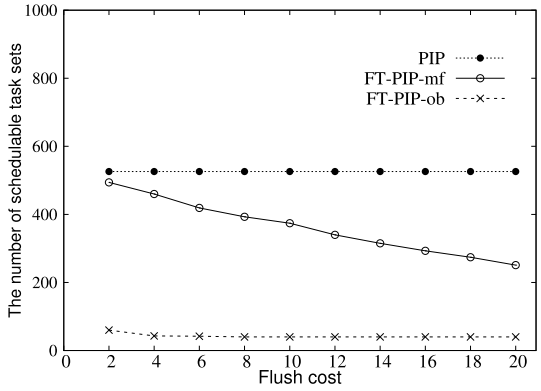
(d) $m = 4, U = 1.2, \gamma = 1, C_{ft}^{x} = 10$



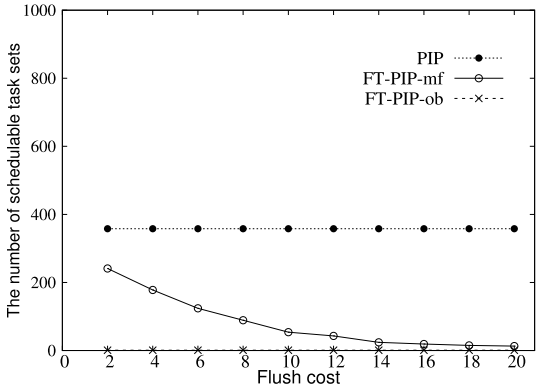
(e) $m = 4, U = 0.8, n = 6, C_{ft}^{x} = 2$



(f) $m = 4, U = 0.8, n = 6, C_{ft}^{x} = 10$



(g) $m = 4, U = 1.2, n = 6, \gamma = 1$



(h) $m = 4, U = 1.2, n = 6, \gamma = 4$

FIGURE 6. Experiment results for $m = 4$.

VIII. RELATED WORK

In literature, many studies considering security for real-time systems were conducted. In [28], Volp *et al.* proposed a modified FP real-time scheduler to prevent information leaks by investigating how much time is required for attackers to access sensitive data on shared resources. They effectively utilized an idle task for preventing attackers from having enough time to examine the resourced shared by real-time tasks. Yoon *et al.* [29] proposed a scheduling randomization protocol for a set of tasks scheduled according to the FP scheduler, which provides obfuscation against timing inference attacks. At each scheduling decision, a budget of allowing priority inversion is calculated, and a job is randomly selected instead of selecting the job with the highest priority. Then, executing such a randomly-selected job is switched to executing the highest-priority job when the budget becomes zero without violating any job deadline.

A notion of FT was first incorporated into the FP non-preemptive scheduler for uniprocessor platforms in [6]. Mohan *et al.* [6] considered a task model that each task has a different security level, and information leakage can happen from a high security-level task to a low security-level one. To judge schedulability of a set of tasks, they transform the problem of bounding the number of invoked FTs during each job's execution to the problem of max-flow. Pellizzoni *et al.* [5] extended this work to preemptive scheduling and a more general task model referred to as the *noleak* model for uniprocessor platforms. Baek *et al.* [7] incorporated the notion of FT into mixed-criticality systems where tasks can have different level of assurance according to its criticality level. However, these studies targeted a uniprocessor platform without considering resource-locking protocols.

The resource-locking protocol has been extensively studied to address the priority inversion problem; by dynamically changing the priority of tasks, the protocol is capable of effectively utilizing limited computing resources [10], [12], [13]. PIP and PCP [10] are the most well-known ones, which promote the priority of tasks to the predefined level to avoid the priority inversion problem on a uniprocessor platform. MPIP [12] and DPCP [13] are variants for a multiprocessor platform, which have been also proposed for various multiprocessor scheduling algorithms such as Pfair scheduling [14], partitioned scheduling [30] and global EDF scheduling [1]. Although the information leakage problem should be addressed for the state-of-the-art real-time systems, these protocols, however, did not consider the security-related constraints.

The scheduling problem for the mixture of dependent tasks and independent tasks has been addressed in various domains. In [36], the non-preemptive scheduling problem of periodic tasks with data dependency for heterogeneous multiprocessor platforms is considered. Data dependency is presented by a notion of directed acyclic graph, and the proposed method exploits linear programming to determine the task schedulability by considering constraints of space, time and precedence relationship. In [37], task partitioning algorithm and schedulability analysis are proposed to determine a strictly periodic task's valid start time and the number of processors required for the given tasks to run without any collision. In [38], mixed integer linear programming is applied to solve the coverage path planning problem that aims at minimizing the completion time of coverage tasks for autonomous heterogeneous unmanned aerial vehicles. However, few of them have addressed security issues.

IX. CONCLUSION

In this paper, we proposed FT-PIP, which incorporates the FT mechanism into multiprocessor real-time scheduling in which mutual exclusion with multiple types of resources is guaranteed under PIP. To this end, we identify a group of delays necessary to enable the FT mechanism, derive their upper bounds, and complete the schedulability analysis for FT-PIP by utilizing the upper bounds. Via simulations, we demonstrated that FT-PIP is effective in satisfying both real-time constraints and security ones on a multiprocessor platform with multiple shared resources. In the future, we would like to extend this work for a more general system model such as mixed-criticality systems. As stated in Section VII, our simulation environment includes a few assumptions such as no preemption or migration cost. It can be a direction of future work to incorporate such cost.

REFERENCES

[1] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
[2] S. Arnon and N. S. Kopeika, "Laser satellite communication network-vibration effect and possible solutions," *Proc. IEEE*, vol. 85, no. 10, pp. 1646–1661, Oct. 1997.
[3] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS)*, Dec. 2007, pp. 149–160.
[4] C. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2019, pp. 90–102.
[5] R. Pellizzoni, N. Paryab, M.-K. Yoon, S. Bak, S. Mohan, and R. B. Bobba, "A generalized model for preventing information leakage in hard real-time systems," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 271–282.
[6] S. Mohan, M. K. Yoon, R. Pellizzoni, and R. Bobba, "Real-time systems security through scheduler constraints," in *Proc. 26th Euromicro Conf. Real-Time Syst.*, Jul. 2014, pp. 129–140.
[7] H. Baek, J. Lee, Y. Lee, and H. Yoon, "Preemptive real-time scheduling incorporating security constraint for cyber physical systems," *IEICE Trans. Inf. Syst.*, vol. E99.D, no. 8, pp. 2121–2130, 2016.
[8] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," *IEEE Trans. Ind. Informat.*, vol. 5, no. 1, pp. 22–37, Feb. 2009.
[9] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, p. 20, Jul. 2007.
[10] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, Sep. 1990.
[11] T. P. Baker, "A stack-based resource allocation policy for realtime processes," in *Proc. 11th Real-Time Syst. Symp.*, 1990, pp. 191–200.
[12] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," in *Proc. 10th Int. Conf. Distrib. Comput. Syst.*, 1990, pp. 116–123.
[13] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *Proc. Real-Time Syst. Symp.*, 1988, pp. 259–269.

[14] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 345–354, 1993.
[15] M. Joseph, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, May 1986.
[16] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1983.
[17] A. Easwaran and B. Andersson, "Resource sharing in global fixed-priority preemptive multiprocessor scheduling," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 377–386.
[18] European Organisation for Civil Aviation Electronics, "Do-178b: Software considerations in airborne systems and equipment certification," EUROCAE, Saint-Denis, France, Tech. Rep., 1992.
[19] N. Grumman. *Replace*. Accessed: May 1, 2021. [Online]. Available: http://www.northropgrumman.com/Capabilities/RePLACE/Pages/default.aspx
[20] N. Grumman. *Reverse Engineering for Large Applications*. Accessed: May 1, 2021. [Online]. Available: http://www.northropgrumman.com/Capabilities/RELA/Pages/default.aspx.
[21] D. Reinhardt, "Certification criteria for emulation technology in the australian defence force military avionics context," in *Proc. 11th Austral. Workshop Saf. Crit. Syst. Softw.*, 2006, pp. 79–92.
[22] H. Baek, H. Lee, H. Lee, J. Lee, and S. Kim, "Improved schedulability analysis for fault-tolerant space-borne sar system," in *Proc. Conf. Korea Inst. Mil. Sci. Technol. (KIIT)*, 2018, pp. 1231–1232.
[23] H. Baek and J. Lee, "Improved schedulability analysis of the contention-free policy for real-time systems," *J. Syst. Softw.*, vol. 154, pp. 112–124, Aug. 2019.
[24] *RTEMS Community, RTEMS Real-Time Operating System*. Accessed: Jun. 4, 2021. [Online]. Available: https://www.rtems.org
[25] *Excalibur Systems, MIL-STD-1553B*. Accessed: Jun. 4, 2021. [Online]. Available: https://www.mil-1553.com
[26] J. Lee, "Time-reversibility for real-time scheduling on multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 1, pp. 230–243, Jan. 2017.
[27] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Proc. 30th IEEE Real-Time Syst. Symp.*, Dec. 2009, pp. 398–409.
[28] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proc. ACM Symp. Inf., Comput. Commun. Secur. (ASIACCS)*, 2008, pp. 44–55.
[29] M.-K. Yoon, S. Mohan, C.-Y. Chen, and L. Sha, "TaskShuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–12.
[30] Y. Oh and S. H. Son, "Allocating fixed-priority periodic tasks on multiprocessor systems," *Real-Time Syst.*, vol. 9, no. 3, pp. 207–239, Nov. 1995.
[31] S. Mohan, M.-K. Yoon, R. Pellizzoni, and R. B. Bobba, "Integrating security constraints into fixed priority real-time schedulers," *Real-Time Syst.*, vol. 52, no. 5, pp. 644–674, Sep. 2016.
[32] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 605–622.
[33] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Proc. Cryptographers Track RSA Conf. Topics Cryptol.*, 2006, pp. 1–20.
[34] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," in *Real-Time Systems*. New York, NY, USA: Springer, 1990, pp. 201–234.
[35] F. Eisenbrand and T. Rothvoß, "Static-priority real-time scheduling: Response time computation is NP-hard," in *Proc. Real-Time Syst. Symp.*, Nov. 2008, pp. 397–406.
[36] J. Chen, C. Du, P. Han, and X. Du, "Work-in-progress: Non-preemptive scheduling of periodic tasks with data dependency upon heterogeneous multiprocessor platforms," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 540–543.
[37] J. Chen, C. Du, F. Xie, and B. Lin, "Scheduling non-preemptive tasks with strict periods in multi-core real-time systems," *J. Syst. Archit.*, vol. 90, pp. 72–84, Oct. 2018.
[38] J. Chen, C. Du, Y. Zhang, P. Han, and W. Wei, "A clustering-based coverage path planning method for autonomous heterogeneous UAVs," *IEEE Trans. Intell. Transp. Syst.*, early access, Mar. 24, 2021, doi: 10.1109/TITS.2021.3066240.

HYEONGBOO BAEK received the B.S. degree in computer science and engineering from Konkuk University, South Korea, in 2010, and the M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2012 and 2016, respectively. He is an Assistant Professor with the Department of Computer Science and Engineering, Incheon National University (INU), South Korea. His research interests include cyber physical systems, real time embedded systems, and system security. He won the Best Paper Award from the 33rd IEEE Real Time Systems Symposium (RTSS), in 2012.

JINKYU LEE (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2004, 2006, and 2011, respectively.

He is an Associate Professor with the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), South Korea, where he joined in 2014. He has been a Research Fellow/Visiting Scholar with the Department of Electrical Engineering and Computer Science, University of Michigan, since 2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems, and cyber-physical systems. He received the Best Student Paper Award from the 17th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), in 2011, and the Best Paper Award from the 33rd IEEE Real Time Systems Symposium (RTSS), in 2012.