Partitioned Real-Time Scheduling for Preventing Information Leakage

DONGHWA KANG\textsuperscript{1}, INHWA JUNG\textsuperscript{1}, KILHO LEE\textsuperscript{2}, AND HYEONGBOO BAEK\textsuperscript{1}\textsuperscript{*}

\textsuperscript{1}Department of Computer Science and Engineering, Incheon National University (INU), Incheon 22012, South Korea
\textsuperscript{2}School of AI Convergence, Soongsil University (SSU), Seoul 06978, South Korea

Corresponding authors: Kilho Lee (khlee.cs@ssu.ac.kr) and Hyeongboo Baek (hbbaek@inu.ac.kr)

This work was supported in part by Incheon National University Research Grant, in 2020; and in part by the National Research Foundation of Korea (NRF) under Grant 2020R1F1A1076425.

ABSTRACT

Recent real-time systems have been gradually connected externally through the Internet. As each subsystem constituting a system has been developed from different vendors, the number of cases of security vulnerabilities in communication between subsystems has increased. In particular, real-time systems are vulnerable to timing inference attacks, such as cache-based side-channel attacks. Among the various methods for handling security problems of these systems, the flush task (FT) method is a simple and effective method to reduce the probability of information leakage by conditionally initializing the state of resources shared by tasks before a given scheduling. However, existing FT methods are for global scheduling only, and techniques for partitioned scheduling have not yet been studied. In this paper, we propose an FT-aware bin-packing (FT-bin-packing) algorithm that effectively allocates real-time tasks to identical multiprocessors to reduce the number of FTs invoked during the schedule of tasks in each processor. Based on the experimental results, we analyze the properties and performance of the FT-bin-packing algorithm. These results indicate that it can improve the performance of existing heuristic algorithms by up to 96.8% depending on the experimental setting.

INDEX TERMS

Real-time system, flush task method, partitioned scheduling.

I. INTRODUCTION

A system that is both functionally and temporally accurate is called a real-time system \cite{1}. For example, an autonomous vehicle repeats three operations: recognition, judgment, and action. Currently, recognition is made using LiDAR, radar, camera, etc., judgment is made using an internal computing system, and action is performed using a motor \cite{2}. These tasks are repeated with a cycle. If each task does not operate within a predefined time, called the deadline, it causes significant human and material damage. Complying with this deadline is called a real-time requirement, and meeting real-time requirements is a crucial concern in developing a real-time system. To satisfy real-time requirements in different fields, the real-time system community has been investigating how to allocate computing resources to real-time tasks and whether each task can complete execution within a deadline. The former method is called a real-time scheduling algorithm, and the latter one is called real-time analysis \cite{3}–\cite{5}.

A sporadic task model has been widely used as a basic model to represent the iterative execution of real-time systems. A sporadic real-time task is expressed in terms of its minimum arrival time (i.e., period), the worst-case execution time, and a deadline. A sporadic task is represented by an infinite sequence of task instances called jobs, with a time constraint of a minimum arrival time interval. Under the constraint of the minimum arrival time interval following the Liu and Layland model \cite{1}, two different jobs must be separated by at least one period. When a job arrives, a task must finish executing within its absolute deadline. Three paradigms are widely used for scheduling real-time tasks on multiprocessor platforms: partitioned, global, and semi-partitioned. A comprehensive survey of multiprocessor scheduling for real-time systems can be found in \cite{6}.

In this study, we consider partitioned scheduling wherein real-time tasks are assigned to identical processors and migration between processors is prohibited \cite{7}. In addition, all tasks are executed with a fixed priority on the processor to which a task is assigned. Unlike global scheduling wherein migration between processors is allowed, partitioned scheduling has the
following advantages. If one task overruns, it affects only the other tasks of the processor to which the task belongs. Because each task is executed only on a single processor, no penalty is applied in terms of migration. For example, if a task is preempted and executed in another processor, the task’s state information and register values (called context) must be set again in the other processor. Hence, communication costs may be incurred, and additional costs, such as cache misses, are required. Partitioned scheduling is simpler in terms of communication than when each processor uses an independent ready queue and when multiple processors use a common queue. In addition, after tasks are assigned to a processor, an advantage is observed in that matured uniprocessor scheduling and analysis techniques can be used.

While many previous studies on real-time systems have focused on real-time requirements, recent studies have handled security issues that can only occur in real-time systems [10]–[12], [15], [16], [21]. This is because, compared to general-purpose systems, a traditional real-time system is recognized as being free from security attacks because it uses a specific network protocol in a limited space where the external connection is blocked. However, recent real-time systems have been gradually connected externally through the Internet. Further, as each subsystem constituting the system has been developed from a different vendor, the number of security vulnerabilities in the communication between subsystems has increased. Thus, information leakage problem has become critical in modern real-time systems because subsystems developed by different vendors share common computing resources. A security attacker can use sensitive information, such as a private key in these shared computing resources, and infers the location of sensitive information through security attacks, such as a side-channel attack.

Among the various methods used to address security problems of such a real-time system, the flush task (FT) method is a simple and effective method to reduce the probability of information leakage by conditionally initializing the state of resources shared by tasks before a given scheduling [13], [16], [21]. This security mechanism inevitably causes additional time overhead. Thus, the scheduling algorithm and schedulability analysis should be designed with this factor being considered. To address this issue, many studies have been conducted on global scheduling. For the first time, a scheduling incorporating the FT mechanism into a fixed-priority (FP) non-preemptive scheduling was proposed [21]. A real-time analysis technique that analyzes whether each task can finish execution within a deadline when tasks are scheduled via this scheduling has also been proposed. The key technique of this method is to find the upper-bound value of FTs that may exist in the interval of interest and to include this in the schedulability analysis. This method was extended to preemptive scheduling in the following work and extended to an exact analysis with a slightly higher computation overhead [15]. In addition, these methods are based on an assumption that computing resources are infinite [16]. In [8], a finite resource was assumed, and a resource-locking protocol considering the critical section problem was proposed. However, these techniques are used for global scheduling, and techniques for partitioned scheduling have not yet been studied. Because a number of real-time systems are implemented with partitioned scheduling, and it is important for these systems to consider their security in that security threats of real-time systems can cause tremendous human and economic loss.

In this study, we aim to develop a new scheduling framework that incorporates a security constraint into existing partitioned scheduling frameworks to address security vulnerabilities caused by the fixed scheduling pattern. Thus, we propose a flush-task-aware bin-packing (hereinafter called FT-bin-packing) algorithm that effectively allocates real-time tasks to identical multiprocessors to reduce the number of FTs invoked during the schedule of tasks in each processor. FT-bin-packing uses a new notion of the security relation graph that visualizes the security relation of real-time tasks associated with a noleak table [16]. Based on the security relation graph, the FT-bin-packing exploits a new task index (used for the task assignment algorithm) re-ordering method to reduce the number of FTs to the extent possible, known as security-relation-aware sorting (SR-SORT). Then, FT-bin-packing is incorporated into the existing bin-packing algorithms [7]. The main idea behind FT-bin-packing algorithm is to split the given tasks into multiple groups so that the number of FTs invoked during scheduling in each group can be reduced. To this ends, the given tasks need to be reordered (via SR-SORT) so that consecutive tasks have no security relation according to the associated noleak table. The security relation graph provides better information with nodes and edges associated with the tasks for the reordering than the noleak table. Then, FT-bin-packing algorithm effectively conducts task partitioning based on SR-SORT and the security relation graph. Therefore, we apply the response time analysis (RTA) [17] that upper bounds the worst-case number of the FTs to guarantee schedulability of a given system. Based on the experimental results, we analyze the properties and performance of the FT-bin-packing algorithm. We found that it can improve the performance of existing heuristic algorithms by up to 96.8% depending on the experimental setting.

In summary, this paper makes the following contributions:

- It proposes a new notion of a security relation graph that visualizes the security relation of the real-time tasks.
- It also proposes a new task-index re-ordering method called SR-SORT.
- A new flush-task-aware bin-packing algorithm, referred to as the FT-bin-packing algorithm, is proposed.
- Extensible experiments are conducted to analyze the properties and performance of the FT-bin-packing algorithm and demonstrate that it improves the performance of existing heuristic algorithms by up to 96.8%.
II. ADVERSARY AND SYSTEM MODEL

In this section, we first present our system model and then our adversary model including attackers’ capabilities and their attack scenarios.

A. SYSTEM MODEL

We consider the Liu and Layland task model [1], [18] in which a set of tasks τ has n sporadic tasks τ₁, · · · , τₙ. Each task τᵢ (1 ≤ i ≤ n) has three parameters, the minimum inter-arrival time Tᵢ, relative deadline Dᵢ, and the worst-case execution time Cᵢ. This indicates that a task τᵢ invokes its instance (called job) infinitely, and each instance should complete its Cᵢ amount of execution before Dᵢ units elapse as a real-time constraint. In addition, the release times of two consecutive jobs are at least Tᵢ apart. We consider a constrained deadline Dᵢ ≤ Tᵢ and assume that τᵢ is scheduled by a given partitioned, non-preemptive, work-conserving scheduling algorithm on identical multiprocessors. That is, each job cannot migrate from one core to another after it is allocated to a processor, each job cannot be preempted by any job after it starts its execution, and each processor is always kept busy provided that an unfinished ready job is available. We consider FP scheduling, which assigns priority to a task rather than a job. This indicates that all jobs released from a higher-priority task equally have higher priorities than those from a lower-priority task. We let hp(k) and hp(k) be a set of tasks whose priorities are lower and higher than τᵢ, respectively. Without loss of generality, we assume quantum-based scheduling, meaning that one value of each parameter indicates a single quantum. We also assume that a smaller index i or task τᵢ is a higher priority for FP scheduling.

Jᵢ and Jᵢ,q denote an arbitrary job and the q-th job for task τᵢ, respectively. Let rᵢ and fᵢ be the release and finish times of a given job Jᵢ. Then, dᵢ = rᵢ + Dᵢ is the absolute deadline of Jᵢ, which means that Jᵢ should finish its execution before dᵢ as a real-time constraint. Job Jᵢ is said to be schedulable when it satisfies the real-time constraint (i.e., fᵢ ≤ dᵢ). Also, a task τᵢ is schedulable when every instance (i.e., job) is schedulable, and a task set τ is said to be schedulable when all tasks of τ are schedulable. Considering the interval between a release time rᵢ and the finish time fᵢ of every job of τᵢ, the response time Rᵢ of a task τᵢ is defined as the longest interval between them. By the definition of Rᵢ, if Rᵢ ≤ Dᵢ holds, then a task set τ is schedulable.

Based on the system model, we define our target problem as follows.

Definition 1 (Problem): Real-time and security constraints should be satisfied under a given partitioned, non-preemptive, work-conserving scheduling algorithm on identical multiprocessors.

Definition 2 (Real-Time Constraint): Every job instance Jᵢ of a task τᵢ should finish its execution before its deadline dᵢ.

Definition 3 (Security Constraint): A timing penalty should be spent between two job executions to cleanse the state of shared resource if it is specified by noleak table.

B. ADVERSARY MODEL

We assume that attackers are aware of the exact parameters of the target task for their security attacks, similar to the general assumptions of existing studies [14], [21], [24]. This assumption stems from the fact that, because the most important functions of a real-time system operate periodically, their behavior can be inferred from long-term observations. In addition, attackers are familiar with the details of the system before performing the attack. An example of many timing inference attacks is a cache-based side-channel attack that exploits the fixed schedule of a real-time system [25], [26]. Under these circumstances, the cache set is associated with the main memory, and the attacker task τₐ hijacked by the attacker runs in parallel with the victim task τ₈. Here, the victim task continuously accesses sensitive information, such as the private key stored in the main memory, and while accessing, the corresponding information is stored in the associated cache. The attacker can repeat the following operation to infer the location of the private key.

τₐ populates some (or all) cache sets with its own data before τ₈ performs its execution (called PRIME operation). After τₐ yields (or completes) execution, it waits for a certain amount of time while τ₈ executes. Then, τₐ conducts the PRIME operation again and measures how long it takes to load data for each cache set. If τ₈ accesses some cache set during its operation, the cache set is replaced with data from τ₈. If so, τ₈ observes access times that are greater (due to cache misses) than the access times of other cache sets that cause cache hits. τ₈ iteratively collects this timing information and infers the location of the private key in main memory.

We aim to reduce the probability of success of such a timing inference attack by selectively initializing the state of a shared resource by referring to the noleak table associated with given real-time tasks. FT-bin-packing utilizing a new schedulability analysis called FT-RTA, partitions tasks into the given processors and then tasks in each processor are scheduled by any existing FP scheduler. Because such the partitioning and analysis are conducted offline in a algorithmic and mathematical way, FT-bin-packing does not cause an additional security vulnerability, other than that the FT scheduling can induce.

III. FT-BIN-PACKING ALGORITHM

In this section, we first introduce notions of the flush task and noleak models. Then, we propose a new task-partitioning algorithm that considers the security constraints.

A. FLUSH TASK WITH NOLEAK MODEL

Among several techniques to prevent information leakage in a real-time system using shared resources, the flush task technique has been widely studied due to the advantage of its simplicity in implementation. The high-level idea of the flush task technique is to initialize the state of the shared resource whenever there is a possibility of information leakage to prevent information from being observed by an attacker.
For instance, when one task finishes its work and successive tasks have a chance to see recently used information remaining in cache or DRAM, both cache and DRAM information can be initialized. An antenna controller software (ACSW) [19], [20] embedded in reconnaissance aerial vehicles can be an example to illustrate the notion of FT. ACSW periodically receives a micro command from a ground station, and the received commands are stored command queue. Then the command is also periodically retrieved to be used to perform the corresponding mission. Here, the information related to the command is very sensitive, and it should be protected from various security attacks. Therefore, the resource shared by relevant tasks should be conditionally flushed when it is necessary. The execution of such a flush task necessarily incurs a temporal overhead, and in a real-time system that considers the flush task, its scheduling algorithm and schedulability analysis must take this into account.

### TABLE 1. Example of noleak relations.

| from | to | τ₁ | τ₂ | τ₃ | τ₄ | τ₅ |
|------|----|----|----|----|----|----|
| τ₁   | -  | T  | F  | T  | F  | F  |
| τ₂   | T  | -  | F  | F  | F  | F  |
| τ₃   | F  | F  | -  | F  | T  | F  |
| τ₄   | F  | F  | F  | -  | F  | F  |
| τ₅   | F  | F  | F  | T  | -  | -  |

Then, when several tasks are executed continuously by a given scheduler, the conditions under which the FT should be executed should be specified at the system design level. Security level and noleak models are the most widely used methods. As the latter is a more general model, this study considers the latter model. Table 1 presents an example of noleak relations associated with five tasks. In Table 1, “T” indicates that the information leakage from the task in the corresponding row to the task in the corresponding column should be prevented, and “F” indicates the opposite one. For example, an FT should be invoked at the end of τ₁’s execution and the start of τ₅’s execution, according to Table 1.

### B. FT-BIN-PACKING ALGORITHM

As our proposed partitioning algorithm is derived from an existing bin-packing algorithm, we first recapitulate the existing algorithm. Then, we present our new heuristic partitioning algorithm that improves the performance of an existing algorithm by recognizing the security relation of given tasks.

The bin-packing algorithm attempts to solve the problem of packing multiple given objects into given bins [7]. In the most basic version, the size of each largest object was at most 1, and the size of each bin was 1. A bin can be filled until the sum of the size(or weight) of objects in the bin is 1 at the maximum. Objects cannot be split and must be allocated according to their size. Previous studies have already demonstrated that packing objects to given bins (called the bin-packing problem) is NP-complete. Therefore, many heuristic algorithms have been proposed to address this problem. The first fit decreasing (FFD) algorithm is one of the simplest algorithms for addressing the bin-packing problem. In FFD, objects are first sorted based on size, and each bin has a certain order. Then, after checking whether the first to last bins can accommodate a given object, the object is packed in the first bin that can accept the object. In case of the problem of allocating real-time tasks into multiple processors, we exploit the bin-packing problem by changing the bins and objects to processors and tasks.

Algorithm 1 includes the FFD algorithm for the task allocation problem. Except for Line 3 from Algorithm 1, it is the same as FFD. FFD considers a set of tasks \( \tau = \{\tau_1, \ldots, \tau_n\} \). Then, for a given task \( \tau_i \) (considered from \( \tau_1 \) to \( \tau_n \)), each processor from \( P_1 \) to \( P_m \) is examined sequentially to verify whether \( \tau_i \) can be allocated to the considered processor without a deadline miss according to a given schedulability analysis (Lines 4–7). Meanwhile, Algorithm 1 uses the flush-task aware response-time analysis (FT-RTA), which is capable of upper bounding the number of FTs invoked during execution of task scheduling and incorporate it into the schedulability judgment. In the procedure, \( \tau_i \) is allocated to the first processor, wherein the included tasks (scheduled by the given scheduling algorithm) make any deadline miss according to the given schedulability analysis (Lines 8–9). If any task cannot be allocated to all processors, then the task set \( \tau \) is deemed not schedulable by the given scheduling algorithm.

A number of other heuristic algorithms exist for task allocation, among which are the best fit decreasing (BFD) and worst fit decreasing (WFD). Similar to FFD, BFD and WFD sort tasks first, in order of highest utilization. When assigning each task to the processor, the considered task is first assigned to the processor with the smallest (i.e., BFD) or largest (i.e., WFD) available (i.e., not assigned to tasks) utilization in the processor, rather than considering the processors in
The underlying idea of FT-bin-packing is to re-assign the index of given tasks so that the noleak relation from node $i$ to node $j$ is called from $i$ to $j$. Because the algorithms have a form similar to that of Algorithm 1, the explanation was not provided.

We present our proposed FT-bin-packing algorithm that utilizes a new notion of the security relation graph described in Figure 1. The noleak relation from any consecutive (in terms of task index) tasks is “F” by utilizing the security relation graph. This concept is based on the fact that the number of FTs invoked in the given interval is proportional to the number of jobs executed in the interval and their noleak relation considered at every iteration of FT-RTA; this is detailed in Section IV-B. The security relation graph for a given task set associated with the corresponding noleak table is constructed as follows: A pair of nodes (called from node $from_i$ and to node $to_i$) for each task $τ_i$ are created, and an directed edge is created from $from_i$ to $to_i$ if noleak relation from $from_i$ to $to_i$ is “F”.

Figure 1 presents a security relation graph associated with the noleak relation in Table 1. As shown in Figure 1, five from and to nodes are created, and an edge between $from_i$ to $to_i$ is created if the corresponding noleak relation is “F”.

Then, the FT-FFD (FT-bin-packing based on FFD) algorithm conducts a task assignment following the steps in Algorithm 1. Before FT-FFD executes steps of FFD (Lines 4–15), it re-orders (i.e., re-assigns) the task indexes of tasks by using the SR-SORT algorithm (Line 3) presented in Algorithm 2. The SR-SORT algorithm first sets $i$ to 1 and $τ_{temp}$ to zero (Line 1). Then, it considers each task $τ_i$ sequentially (i.e., from $i == 1$ to $i == n$), and if it is not an element of $τ_{temp}$ (Lines 2–6). Thereafter, for a task $τ_j$ (after $τ_j = τ_j$ is conducted in Line 3), each task $τ_k$ is considered (Lines 8–9), and then $τ_k$ moves to $τ_{temp}$ if noleak($τ$, $τ_j$, $τ_k$) is F (i.e., the noleak relation from $τ_j$ to $τ_k$ is “F”) and $τ_k$ is hold (Lines 10–14). Here, $τ_j$ becomes $τ_k$ if $τ_j$ moves to $τ_{temp}$ during the above iteration (Line 13). Each task index $i$ of $τ_{temp}$ is newly assigned from 1 to $n$ according to the inserted order after all iterations (Line 18), and then return $τ_{temp}$ (Line 19). $τ_{temp}$ will be used instead of $τ$ (Line 3 in Algorithm 1), to conduct the remaining steps (i.e., Lines 4–15 in Algorithm 1) of the FT-FFD algorithm.

\begin{algorithm}[h]
\caption{SR-SORT Algorithm}
\begin{algorithmic}[1]
\STATE $i \leftarrow 1$; $τ_{temp} = \emptyset$
\WHILE{$i \leq n$}
\STATE $j \leftarrow i$; $k \leftarrow 1$
\IF{$τ_k \notin τ_{temp}$}
\STATE $τ_{temp} = τ_{temp} \cup τ_k$
\STATE $τ = τ \setminus τ_i$
\STATE $τ = τ \setminus τ_k$
\STATE $j \leftarrow k$
\ENDIF
\WHILE{$j \leq n$}
\WHILE{$k \leq n$}
\IF{noleak($τ$, $τ_j$, $τ_k$) = F and $τ_k \notin τ_{temp}$}
\STATE $τ_{temp} = τ_{temp} \cup τ_k$
\STATE $τ = τ \setminus τ_k$
\STATE $j \leftarrow k$
\ENDIF
\ENDIF
\ENDWHILE
\end{algorithmic}
\end{algorithm}

Each task index $i$ of $τ_{temp}$ is newly assigned from 1 to $n$ according to the inserted order.

\begin{algorithm*}[h]
\begin{algorithmic}[1]
\STATE $τ_{temp}$
\end{algorithmic}
\end{algorithm*}

IV. SCHEDULABILITY ANALYSIS

As presented in the previous section, our proposed partitioning framework utilizes FT-RTA for the uniprocessor to judge the schedulability of the given tasks (in Line 7 of Algorithm 1). This section describes the procedure of FT-RTA, including the method for bounding the number of FT tasks invoked during job execution.

A. SCHEDULABILITY ANALYSIS

FT-RTA attempts to find the worst-case response time $R_k$ of a given task $τ_k$ associated with FTs and to verify whether it holds $R_k \leq D_k$. If it holds, then every job of $τ_k$ is guaranteed to finish its execution without a deadline miss.

Unlike preemptive scheduling, it should consider so-called “self-pushing phenomenon” that occurs because execution of higher-priority jobs released during non-preemptive execution of a previous job of $τ_k$ are delayed, thereby inducing a higher interference to current job of $τ_k$. This implies that the worst-case response time of $τ_k$ cannot be derived by investigating only the first job in the critical instance [17], but multiple jobs should be examined until the execution of tasks with a priority higher than or equal to $τ_k$’s priority is finished. FT-RTA exploits the existing schedulability analysis considering the self-pushing phenomenon proposed in [39], and then incorporates the method to upper-bound the number of FTs that can be invoked during the execution of involved jobs into the analysis.

To this ends, FT-RTA first computes level-$k$ active period, defined as follows.

Definition 4 (Level-$k$ Active Period [39]): A level-$k$ active period is an interval [$a$, $b$] such that a processor is busy due to remaining execution of jobs of tasks whose priority is higher than or equal to $τ_k$, and the processor is not in the situation in $a$ and $b$.

Then, level-$k$ active period can be derived by the following

$$\ell_k^{(0)} = B_k + C_k + C_{ft},$$

$$\ell_k^{(x)} = B_k + \sum_{τ_i \in hp(k)} \left(\frac{\ell_k^{(x-1)}}{T_i}\right) C_i + FT_k(\ell_i^{(x-1)} \mid τ_i \in hp(k), 1),$$

where $B_k$ is the execution time of the $k$th level, $C_k$ is the setup time of the $k$th level, and $C_{ft}$ is the execution time of an FT.
where $B_k$ is the maximum blocking time that can result from a lower-priority task calculated by

$$B_k = \max_{y \in hp(k)} C_y - 1, \quad (2)$$

and $I_i$ is the number of jobs calculated by

$$I_i^{(x-1)} = \left[ \frac{\ell_k^{(x-1)}}{T_i} \right], \quad (3)$$

and $FT_k(I_i^{(x-1)}|\tau_i \in hp(k), 1)$ is the maximum number of FTs invoked during the execution of a job of $\tau_k$ in $\ell_k^{(x-1)}$, which is detailed in the next section. Let $\ell_k$ be the smallest value for which $\ell_k = \ell_k^{(x-1)}$. Then, $R_i$ is derived by calculating the response times of all jobs $J_{k,q}$, with $q \in [1, Q_i]$, where $Q_i$ is computed by

$$Q_i = \left\lceil \frac{\ell_k}{T_k} \right\rceil. \quad (4)$$

Then, the start time $s_{k,q}$ of a generic job $J_{k,q}$ can be computed considering the blocking time $B_k$, the execution times of $(q-1)$ prior jobs of $\tau_k$, interference from higher-priority jobs, and the upper-bounded number of FTs invoked between their executions. The following derives $s_{k,q}$ when $s_{k,q} = s_{k,q}^{(y)}$,

$$s_{k,q}^{(0)} = B_k + \sum_{\tau_i \in hp(k)} C_i + FT_k(I_i^{(x-1)} = 1|\tau_i \in hp(k)), \quad (5)$$

$$s_{k,q}^{(y)} = B_k + (q-1)C_k + \sum_{\tau_i \in hp(k)} \left( \frac{s_{k,q}^{(y-1)}}{T_k} \right) + 1)C_i + FT_k(I_i^{(y-1)}|\tau_i \in hp(k), q), \quad (5)$$

where $I_i^{(y-1)}$ is the number of jobs calculated by

$$I_i^{(y-1)} = \left[ \frac{s_{k,q}^{(y-1)}}{T_k} \right] + 1, \quad (6)$$

and $FT_k(I_i^{(y-1)}|\tau_i \in hp(k), q)$ is the maximum number of FTs invoked during the execution of $q$ jobs of $\tau_i$ in $s_{k,q}^{(y-1)}$. Once $s_{k,q}$ is derived, the finish time $f_{k,q}$ for $J_{k,q}$ is directly calculated by

$$f_{k,q} = s_{k,q} + C_k. \quad (7)$$

Also, $R_k$ for $\tau_k$ can be calculated by

$$R_k = \max_{q \in [1, Q_i]} \{f_{k,q} - (q-1)T_k\}. \quad (8)$$

Finally, the given task set is sufficiently schedulable if

$$\forall k = 1, \ldots, n \ R_k \leq D_k. \quad (9)$$

B. FT BOUNDS

In the previous section, as the underlying concept of FT-RTA, we examined how it judges the schedulability of $\tau_k$ through iteration. As the last part of FT-RTA, we examine in this section how to upper-bound the number of flush tasks that can occur while the job of $\tau_k$ finishes executing. Through some observations, we can replace this problem with the well-known max-flow problem [9] and present a solution.

Recall that an FT is invoked between two tasks $\tau_a$ and $\tau_b$, and the noleak relation from $\tau_a$ to $\tau_b$ is “T”. This reasoning implies that the number of FTs invoked during the execution $\tau_k$ in an interval is determined by the (i) the number of job invocations, (ii) execution sequence and (iii) noleak relations associated with the given noleak table. To address the problem considering (i), (ii), and (iii) simultaneously, we transform our problem to a well-known max-flow problem by exploiting a notion of the noleak graph. Noleak graph consists of (i) pairs of nodes and few edges (connected between nodes) associated with (ii) and (iii). The nodes and edges of noleak graph corresponding to $FT_k(I_i^{(y-1)}|\tau_i \in hp(k), q)$ is constructed with the following rules:

N1. $I_i^{(x-1)}$ (or $I_i^{(y-1)}$) send nodes (denoted by send$_{i,1}$, send$_{i,2}$, ..., send$_{i,q}$, respectively) and $I_i^{(x-1)}$ (or $I_i^{(y-1)}$) receive nodes (denoted by recv$_{i,1}$, recv$_{i,2}$, ..., recv$_{i,q}$, respectively) are created.

N2. $q$ (or a single) send node(s) (denoted by send$_{i,1}$, send$_{i,2}$, ..., send$_{i,q}$) and $q$ (or a single) receive nodes (denoted by recv$_{i,k}$, recv$_{i,2}$, ..., recv$_{i,q}$) are created.

Then, the followings specify the constraints of directed and FT edges of a noleak graph.

E1. A directed edge from an arbitrary send node send$_a$ to an arbitrary send node recv$_b$ is created if noleak relation from $\tau_a$ to $\tau_b$ is “T”.

FIGURE 2. Example of the noleak graph corresponding to Table 2.
E2. A send node send_a has at most one FT edge.

E3. A receive node recv_b has at most one FT edge.

We now show that noleak graph corresponding to
\( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) (likewise \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), q) \)) constructed by the above rules can upper-bound the number of FTs invoked between a blocking job (whose execution is upper-bounded by \( B_k \)), \( I^{(x-1)}_k \) (likewise \( I^{(y-1)}_k \)) jobs of \( \tau_i \in hp(k) \), and a (likewise \( q \)) job(s) of interest \( J_k \). To this ends, we first define valid job sequence of \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) and \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), q) \) as follows.

Definition 5: (Valid Job Sequence for \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \)): A job sequence for \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) is valid if it starts with any job in \( lp(k) \), and the remaining consists of \( I^{(x-1)}_k \) higher-priority jobs \( J_k \) of \( \tau_i \in hp(k) \) and a job \( J_k \) of \( \tau_k \).

Definition 6: (Valid Job Sequence for \( FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \)): A job sequence for \( FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \) is valid if it starts with any job in \( lp(k) \), and the remaining consists of \( I^{(y-1)}_k \) higher-priority jobs \( J_k \) of \( \tau_i \in hp(k) \) and \( (q-1) \) jobs \( J_k \) of \( \tau_k \).

Note that there are multiple valid sequences for the function \( FT_k(\cdot) \) according to the definitions. Then, we also define valid flow assignment as follows.

Definition 7 (Valid Flow Assignment): A flow assignment of a given noleak graph is valid if it satisfies N1, N2, E1, E2, and E3.

Let \( \lambda \) be the number of FTs invoked jobs in a valid job sequence for \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) or \( FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \), and \( F \) be the flow value of the corresponding noleak graph. The following lemmas prove that a valid sequence with \( \lambda \) results in the valid flow assignment with \( F + 1 \).

Lemma 1: For any valid job sequence for \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) in which the number of invoked FTs is \( \lambda \), there exists a valid flow assignment with the corresponding noleak graph of which flow value \( F \) satisfies \( \lambda \leq F \).

Proof: Assume that there is a valid job sequence under which \( \lambda \) FTs are invoked. We classify jobs in the valid job sequence into the three groups; the first job \( J_f \) is \( lp(k) \) if any, the last job \( J_l \), and middle jobs that are the remaining jobs except the first and last jobs. Note that \( J_f \) does not exist if \( \tau_k \) is the lowest priority task. By N1 and N2, a send and a recv node of every job except \( J_f \) in the valid job sequence are created.

We first consider the following four cases:

i) An FT is invoked between \( J_f \) and the first job \( J_a \) of middle jobs if noleak relation from \( \tau_f \) to \( \tau_a \) is “T”. Note that noleak graph does not consider the first job \( J_f \) and its associated FT, there is no action in noleak graph.

ii) Then, a single FT is invoked between any consecutive two jobs, \( J_a \) and \( J_b \), in middle jobs only when noleak relation from \( \tau_a \) to \( \tau_b \) is “T”. By E1, a directed edge from \( \text{send}_a \) to \( \text{recv}_b \) is created for every \( J_a \) that satisfies noleak relation from \( \tau_a \) to \( \tau_b \) is “T”. The mechanism of FT invocation, if an FT is invoked between \( J_a \) and \( J_b \), an FT, and \( J_b \) are executed consecutively), then another FT is cannot be invoked between \( J_a \) and \( J_b \). Such the constraint is realized by E2 and E3.

iii) For the last job \( J_l \), an FT is invoked between the last job \( J_b \) of middle jobs and \( J_l \) if noleak relation from \( \tau_b \) to \( \tau_l \) is “T”.

iv) We do not consider an FT after \( J_l \), but an FT edge from \( \text{send}_r \) to a certain node \( \text{recv}_b \) is created. This case can happen because noleak graph does not recognize the job sequence of middle and last jobs in a valid job sequence, and thus a send node \( \text{send}_r \) is created.

For i), \( F \) does not increase even if an FT is invoked because noleak graph does not consider the first job \( J_f \) and its associated FT. For ii), \( F \) increases by one whenever an FT is invoked. For iii), \( F \) also increases by one if an FT is invoked. For iv), \( F \) can increase even if an FT after \( J_l \) is ignored.

Considering all cases, \( \lambda \leq F + 1 \) holds; \( \lambda \leq F + 1 \) happens in all cases, and \( F \) occurs while i) and iv) occur selectively.

Therefore, the lemma hold.

Lemma 2: For any valid sequence for \( FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \) in which the number of invoked FTs are \( \lambda \), there exists a valid flow assignment with the corresponding noleak graph of which flow value \( \lambda = F + 1 \).

Proof: The proof follows Lemma 1.

Theorem 1: Let \( F^* \) be the max-flow value of a noleak graph corresponding to \( FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \) or \( FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \), and then

\[
FT_k(I^{(x-1)}_k|\tau_i \in hp(k), 1) \\
\text{or} FT_k(I^{(y-1)}_k|\tau_i \in hp(k), q) \leq F^* + 1.
\]

Proof: The theorem holds by Lemmas 1 and 2.

**TABLE 2. Example of noleak relations.**

| from | to  | \( \tau_1 \) | \( \tau_2 \) | \( \tau_3 \) |
|------|-----|-------------|-------------|-------------|
| \( \tau_1 \) | -   | F           | T           |
| \( \tau_2 \) | T   | -           | F           |
| \( \tau_3 \) | T   | F           | -           |

Based on the construction rule and the associated constraints, \( FT_k(L) \) derives the upper-bounded number of FTs in an interval of length \( L \) (as incorporated in Equations (1) and (5), respectively) as described in Figure 2. Figure 2 presents an example of noleak graph corresponding to Table 2. The dotted lines in Figure 2 represent directed edges, and solid lines means FT edges. Also, a circle drawn with a solid line is a send node, and that with a dotted line is a receive node. Finally, the number in each circle is task index. Figure 2 implies that \( \tau_1 \), \( \tau_2 \), and \( \tau_3 \) each invokes (at most) two jobs in an interval of length \( L \). Under the constraints of directed and FT edges of the noleak graph, there can be at most four FT edges, which can be derived by any well-known solution for max-flow problem such as the Ford-Fulkerson algorithm [9].
C. TIME COMPLEXITY

Our proposed FT-bin-packing first re-orders the priorities of the given tasks according to SR-SORT, and then it assigns the tasks into processors based on the considered bin-packing algorithm (e.g., FT-FFD when it comes to the FFD algorithm) by utilizing FT-RTA in conjunction with a mechanism to upper-bound the number of FTs that can be invoked. As shown in Algorithm 2, the time complexity of SR-SORT is \( O(N^3) \) where \( N \) is the number of tasks as it is conducted with a triple loop operation. Then, our considered FT-FFD tests the schedulability via FT-RTA of tasks in a processor whenever a task is inserted to the processor, thereby resulting in \( O(N \cdot M) \) multiplying the time complexity of FT-RTA, where \( M \) is the number of processors. As shown in Section IV, FT-RTA first derives the level-\( k \) active period \( \ell_k \) for each \( t_k \), each whose length can be the least comment multiple of all periods of tasks in the worst-case. Then, the response time of all jobs of \( t_k \) in \( \ell_k \) is derived. The response time of each job in \( \ell_k \) is calculated by adding \( C_k \), \( B_k \), execution times of its higher-priority tasks, and the upper-bounded number of FTs computed by \( FT_k() \). \( FT_k() \) is calculated by \( O(N^3) \) with the any existing max-flow algorithm. Thus, the time complexity of FT-RTA is \( O(N \cdot LCM \cdot N^2 \cdot N^3) = O(LCM \cdot N^6) \). Therefore, the total time complexity is \( O(N^3) + O(N \cdot M \cdot LCM \cdot N^5) = O(M \cdot LCM \cdot N^7) \).

V. EVALUATION

In this section, we evaluate the effectiveness of FT-bin-packing under varying system parameters in our experimental environment. To this end, we randomly generate a number of task sets according to a well-known task-set generation method referred to as UUnifast-discard [22]. 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 \( U \) \((0.1m, 0.2m, 0.3m, 0.4m, 0.5m, 0.6m) \) that is defined as \( \sum_{t \in T} C_i / T_i \). For a given task utilization for \( t_i \) (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 also consider (iv) the flush cost \( C_f \) \((4, 8, 12, \cdots, 36, 40)\), defined as the time required to initiate the status of the shared resource. Next, each element of the \( n \)-by-\( n \) noleak table for each resource \( \lambda_i \) is randomly set to either 0 or 1. For each combination of input parameters (i)–(iv), we generate 10,000 task sets, thereby resulting in \( 2 \cdot 7 \cdot 6 \cdot 11 = 9, 240, 000 \) task sets in total. As a performance metric, we measure the number of task sets deemed schedulable by the proposed heuristic algorithm 1 utilizing FT-RTA.

As our proposed approach supports any FP scheduling with its corresponding schedulability analysis for each processor, we consider the rate-monotonic (RM) scheduling algorithm (with FT-RTA) because of it is one of the optimal one for uniprocessor platforms. We consider the following heuristic algorithms for the performance comparison.

- **FF**: The FFD algorithm described in 1, except for Line 3 of 1.
- **WF**: The WFD algorithm introduced in Section III-B, which is similar to FF.
- **FT-FF**: The FT-FFD algorithm presented in 1.
- **FT-WF**: The SR-SORT algorithm (i.e., Algorithm 2) incorporated WFD algorithm; that is, SR-SORT is conducted before WFD is done.
- **FT-FF+WF**: The algorithm that returns “Success” if either FT-FF or FF successfully assigns all the tasks of the given task set to the given processors.
- **FT-WF+WF**: The algorithm that returns “Success” if either FT-WF or WF successfully assigns all the tasks of the given task set to the given processors.

Note that we do not consider the BFD algorithm for this section because it exhibits a similar performance with the FFD algorithm under our experiment environment.

Figures 3(a)-(f) plot the number of schedulable task sets under the considered heuristic algorithms for \( m = 4 \) with varying values of \( U, n, \) and \( C_f \). Figures 3(a) and (b) show the number of schedulable task sets according to changing value of \( U \) for a given values of \( m, n, \) and \( C_f; m = 4 \) and \( n = 6 \) are common, and \( C_f \) is 12 for Figure 3(a) and 40 for Figures 3(b). Figures 3(c) and (d) address the performance of the considered algorithms under the different number of tasks in a task set for a given value of \( m, U, \) and \( C_f; \) the parameters are \( m = 4, U = 2.4, \) and \( C_f = 4 \) for Figures 3(c), and \( m = 4, U = 1.2, \) and \( C_f = 16 \). Finally, Figures 3(e) and (f) plot the number of schedulable task sets for varying flush costs (i.e., \( C_f \)) for \( m = 4, U = 2.4, \) and two \( C_f \) values, 5 and 10 for Figures 3(e) and (f), respectively. Note that FT-FF+WF, FT-WF, and FT-FF+FF in Figure 3(e) exhibits very similar performance, and thus lines of FT-FF and FT-FF+FF are obscured by FT-FF+FF’s one.

From Figures 3(a)-(f), we have the following observations.

**O1.** Our proposed methods improve the performance of existing heuristic algorithms, FF and WF (in Figures 3(a)-(f)).

**O2.** A larger performance improvement is achieved when our method is applied to WF rather than the case of FF in most cases (in Figures 3(a)-(f)).

**O3.** The increasing values of \( U, n, \) and \( C_f \) lower the performance of all the considered algorithms (in Figures 3(a)-(b)).

**O4.** Despite O3, it increases the ratio between the number of schedulable task sets under the FT- series and that of the existing heuristic algorithms, FF and WF in most cases (in Figures 3(c)-(d), and Figures 3(e) vs. (f)).

Here, O1 demonstrates the effectiveness of our proposed method in assigning real-time tasks considering security constraints. As introduced in Section III-B, the underlying idea of FT-bin-packing is to re-assign the index of given tasks so that the noleak relation of any consecutive (in terms of task index) tasks is “F” by utilizing the security relation graph, and this idea is based on the fact that the number of FTs upper-bounded by the method in Section IV-B is proportional to the
number of jobs executed in an interval of length $\ell_k$ and their noleak relation considered at every iteration of the utilized FT-RTA.

As explained in Section III-B, WF assigns a given task to a processor with the largest remaining utilization among all the considered processors. This approach does not effectively use the full utilization of the processor because it results in larger fragmentation (i.e., available utilization) after a task is assigned. When the security constraint is considered, such a limitation is more pronounced, resulting in a larger performance gap between FF and WF, as shown in Figures 3(a)–(f). We can observe in O2 that our proposed methods compensate for the limitation of WF by effectively reordering the tasks by considering the given noleak table.

Increasing the values of $U$, $n$, and $C_{ft}$ lead to O2 for different reasons. As the $U$ value (fixed $n$) increases, the utilization of each task in the task set increases, indicating that the $C_i$ value of each task generally increases. This implies that the interference received from high priority also increases naturally from the standpoint of $\tau_k$, which is the target of schedulability determination, and the value of $R_k$ increases with high probability. Second, the larger the value of $n$ (using
the fixed value \( U \), the fewer the number of schedulable task sets. This is because when estimating the maximum execution time of \( \tau_i \in hp(k) \) in the interval of \( \ell_k \) length, the schedulability test uses the upper-bound value based on the worst-case scenario, rather than the exact value. Therefore, as the number of tasks in the working set increases, the pessimistic calculation of interference becomes more pronounced. Finally, the increased value of \( C_f \) definitely increases the worst-case response time of each operation when the upper bounded FT number is not changed, reducing the feasibility of just-in-time execution.

Although the increasing values of \( U \), \( n \), and \( C_f \) lower the performance of all the considered algorithms (as observed in \( O3 \)), it increases the ratio between the number of schedulable task sets under the FT series and that of existing heuristic algorithms, as observed in \( O4 \). This implies that our proposed technique exhibits a high percentage of performance improvement for a set of tasks that are difficult to schedule. For example, 575 and 1132 task sets are schedulable for \( m = 4 \), \( U = 2.4 \), \( n = 5 \), and \( C_f = 40 \) under WF and FT-WF+WF, respectively, thereby resulting in 96.8% performance improvement.

VI. RELATED WORK

Owing to its low runtime overheads and simplicity in scheduling, partitioned scheduling has been widely used in multiprocessor real-time systems and extensively studied [27]–[32]. As optimal task partitioning is known to be NP-hard (even harder when it comes to the problem considering few resources), an integer linear programming (ILP) method for an optimal resource-aware task assignment was proposed although it is computationally intractable [27]. In [28], Baruah has studied the task partitioning problem on a heterogeneous multiprocessor platform and designed a polynomial heuristic algorithm to aim at near-minimizing makespan of task executions. In [29], a well-known semi-partitioned method referred to as C = D scheduling was revised to be enabled for dynamic workloads consisting of reservation servers. The proposed method was improved to perform in linear time, and thus the computational complexity is dramatically reduced compared to the existing state-of-the-art method. Data inconsistency in shared resources was also addressed by Al-Bayati et al. in [30] as they proposed a new resource-locking protocol called communication affinity and slack with retries (CASA) which outperforms the existing well-known two protocols, the multiprocessor priority ceiling protocol (MPCP) and the multiprocessor stack resource policy (MSRP). Additionally, partitioned scheduling problems in real-time distributed systems and associated load-balancing issues are addressed [31], [32].

Non-preemptive scheduling prevents running tasks from being preempted until the (running) task is completed, and it is suitable for systems where scheduling cost (e.g., preemption or migration) is extremely high or preemption is inherently prohibited. Due to its advantages, there were a number of studies to develop non-preemptive scheduling for various real-time systems [33]–[36]. In [33] targeted controller area network (CAN) and proposed a revised response-time analysis by revealing the existing schedulability analysis’s flaw in optimality. Nasri and Brandenburg proposed an exact sustainable schedulability test for the fixed-priority non-preemptive scheduling supporting both work-conserving and non-work-conserving schedulers in [34]. They also proposed online scheduling policy based on the pre-constructed offline table by addressing the problem of non-preemptive scheduling periodic tasks on resource-constrained platform [35]. This work is motivated by a fact that the existing fully online scheduling methods provide the low performance in schedulability analysis, and the fully offline approach requires too much memory. In [36], parallel task models for partitioned and non-preemptive on multiprocessors are considered, and a new schedulability analysis for DAG (directed acyclic graphs) was proposed.

Recently, as security issues for real-time systems have emerged as a hot topic, many studies considering security in the real-time system field have been conducted. In [23], Volp et al. proposed a new FP real-time scheduler to reduce the probability of information leakage by investigating the time required for attackers to access shared sensitive data. They use idle tasks efficiently to prevent attackers from having sufficient time to access resources shared by real-time tasks. Yoon et al. [24] proposed a scheduling randomization protocol for tasks scheduled by the FP scheduler, which can reduce the possibility of timing inference attacks. In their proposed method, the priority inversion budget that one task can have is calculated for every scheduling decision, and a random task is executed, rather than the highest priority being scheduled by that amount. Then, such random scheduling returns to the original given FP scheduling without a task deadline miss when the given priority inversion budget becomes zero. The FT technique was first incorporated into the FP non-preemptive scheduler for the uniprocessor platform in [21]. The first study assumes that every task has its own security level, which forces FTs to be conditionally invoked, considering the different security levels of tasks. To ensure the schedulability of tasks under the assumption, the idea of transforming the problem of bounding the number of FTs that can occur while each task is running into a max-flow problem has been proposed. Pellizzoni et al. [15] extended this study to a more general model called the noleak model and proposed a new FT method applicable to preemptive scheduling. Baek et al. [16] incorporated the notion of FT into mixed-criticality systems in which tasks have different levels of criticality. In [37], a new security overload model was proposed to quantify overheads of security tasks such as confidentiality, integrity, and authentication conducted in real-time systems, and a new EDF (earliest deadline first) scheduling considering the overheads was proposed. In [38], an EDF schedulability driven security optimization method for real-time systems was proposed. Unlike the existing approaches, we consider a problem of partitioned and non-preemptive scheduling, and propose a new heuristic
bin-packing algorithm and schedulability analysis. Also, the existing studies considering FT mechanisms is limited to the uniprocessor platforms, while our work is applicable to multiprocessor platforms.

VII. CONCLUSION
In this paper, we proposed a new scheduling framework, called FT-bin-packing, that incorporates a security constraint into existing partitioned scheduling, such as FFĐ and WFD, to address security vulnerabilities caused by the fixed scheduling pattern of real-time systems. FT-bin-packing uses a new notion of the security relation graph and SR-SORT to visualize the security relation of real-time tasks associated with a noleak table and to reduce the number of FTs to the extent possible; then, the FT-bin-packing where FT-RTA is integrated into existing bin-packing algorithms. Based on the experimental results, we analyzed the properties and performance of the FT-bin-packing algorithm. These results indicate that the algorithm can improve the performance of existing heuristic algorithms by up to 96.8% depending on the experimental setting. In future work, we will attempt to incorporate another security mechanism, such as TaskShuffler [24], into identical multiprocessor platforms.

REFERENCES
[1] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” J. ACM, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[2] J. Wang, J. Liu, and N. Kato, “Networking and communications in autonomous driving,” IEEE Commun. Surveys Tuts., vol. 21, no. 2, pp. 1243–1274, Dec. 2019.
[3] B. B. Brandenburg and M. Gul, “Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations,” in Proc. IEEE Real-Time Syst. Symp. (RTSS), Nov. 2016, pp. 1–15.
[4] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo, “Schedulability analysis of conditional parallel task graphs in multicore systems,” IEEE Trans. Comput., vol. 66, no. 2, pp. 339–353, Jun. 2016.
[5] A. Biondi, G. C. Buttazzo, and M. Bertogna, “Schedulability analysis of hierarchical real-time systems under shared resources,” IEEE Trans. Comput., vol. 65, no. 5, pp. 1593–1605, May 2016.
[6] R. I. Davis and A. Burns, “Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems,” Real-Time Syst., vol. 47, no. 1, pp. 1–40, Jan. 2011.
[7] D. D. Niz and R. Rajkumar, “Partitioning bin-packing algorithms for distributed real-time systems,” Int. J. Embedded Syst., vol. 2, no. 3/4, p. 196, 2006.
[8] H. Baek and J. Lee, “FT-PIP: Flush task incorporated priority-inheritance protocol to reduce information leakage on multiprocessor real-time systems,” IEEE Access, vol. 9, pp. 81882–81894, 2021.
[9] W. L. Wilkinson, “An algorithm for universal maximal dynamic flows in a network,” Oper. Res., vol. 19, no. 7, pp. 1602–1612, Dec. 1971.
[10] J. Son and J. Alves-Foss, “Covert timing channel analysis of rate monotonic real-time scheduling algorithm in MLS systems,” in Proc. IEEE Inf. Assurance Workshop, Jun. 2006, pp. 361–368.
[11] M.-K. Yoon, S. Mohan, J. Choi, and I. Sha, “Memory heat map: Anomaly detection in real-time embedded systems using memory behavior,” in Proc. 52nd Annul. Design Automat. Conf., Jun. 2015, pp. 1–6.
[12] M. M. Zadeh, M. Salem, N. Kumar, G. Cutulenco, and S. Fischmeister, “SiPTA: Signal processing for trace-based anomaly detection,” in Proc. 14th Int. Conf. Embedded Softw. (EMSOFT), 2014, pp. 1–10.
[13] H. Baek and J. Lee, “Incorporating security constraints into mixed-criticality real-time scheduling,” IEICE Trans. Inf. Syst., vol. 100, no. 9, pp. 2068–2080, 2017.
[14] C.-Y. 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.
[15] R. Pellizzoni, N. Payab, 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. (RTAS), Apr. 2015, pp. 271–278.
[16] 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. 99, no. 8, pp. 2121–2130, 2016.
[17] M. Joseph, “Finding response times in a real-time system,” Comput. J., vol. 29, no. 5, pp. 390–395, May 1986.
[18] A. Mok, “Fundamental design problems of distributed systems for the hard-real-time environment,” M.S. thesis, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1983.
[19] 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. Sys. Technol. (KIST), 2018, pp. 1231–1232.
[20] 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.
[21] 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.
[22] 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.
[23] M. Völp, C. J. Hamann, and H. Härtig, “Avoiding timing channels in fixed-priority schedulers,” in Proc. ACM Symp. Inf., Comput. Commun. Secur., 2008, pp. 44–55.
[24] 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.
[25] F. Liu, Y. Yaron, 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.
[26] 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. 1–25.
[27] A. Wieder and B. B. Brandenburg, “Efficient partitioning of sporadic real-time tasks with shared resources and spin locks,” in Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES), Jun. 2013, pp. 49–58.
[28] B. S. K. Baruah, “Partitioning real-time tasks among heterogeneous multiprocessors,” in Proc. Int. Conf. Parallel Process., 2004, pp. 467–474.
[29] D. Casini, A. Biondi, and G. C. Buttazzo, “Task splitting and load balancing of dynamic real-time workloads for semi-partitioned EDF,” IEEE Trans. Comput., vol. 70, no. 7, pp. 1618–1811, Nov. 2020.
[30] Z. Al-bayati, Y. Sun, H. Zeng, M. D. Natale, Q. Zhu, and B. H. Meyer, “Partitioning and selection of data consistency mechanisms for multicore real-time systems,” ACM Trans. Embedded Comput. Syst., vol. 18, no. 4, pp. 1–28, Jul. 2019.
[31] A. Mehiawia, E. Wozniak, S.ucci-Piergiovanni, C. Mraidha, M. Di Natale, H. Zeng, J.-P. Babau, L. Lemarchand, and S. Gerard, “A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems,” in Proc. 14th ACM SIGPLAN/SIGBED Conf. Lang., Compil. Tools Embedded Syst. Jun., 2013, pp. 121–132.
[32] F. D. Casini, A. Biondi, and G. Buttazzo, “Semipartitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing,” in Proc. 29th Euromicro Conf. Real-Time Syst. (ECRTS), 2017, pp. 28–30.
[33] R. I. Davis, A. Burns, R. J. Brit, and J. J. Lukkien, “Controller area network (CAN) schedulability analysis: Refuted, revisited and revised,” Real-Time Syst., vol. 35, no. 3, pp. 239–272, Apr. 2007.
[34] M. Nasri and B. B. Brandenburg, “An exact and sustainable analysis of non-preemptive scheduling,” in Proc. IEEE Real-Time Syst. Symp. (RTSS), Dec. 2017, pp. 12–23.
[35] M. Nasri and B. B. Brandenburg, “Offline equivalence: A non-preemptive scheduling technique for resource-constrained embedded real-time systems (outstanding paper),” in Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS), Apr. 2017, pp. 75–86.
D. Casini, A. Biondi, G. Nelissen, and G. Buttazzo, “Partitioned fixed-priority scheduling of parallel tasks without preemptions,” in Proc. IEEE Real-Time Syst. Symp. (RTSS), Dec. 2018, pp. 11–14.

T. Xie and X. Qin, “Scheduling security-critical real-time applications on clusters,” IEEE Trans. Comput., vol. 55, no. 7, pp. 864–879, Jul. 2006.

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.

G. C. Buttazzo, M. Bertogna, and G. Yao, “Limited preemptive scheduling for real-time Systems. A survey,” IEEE Trans. Ind. Informat., vol. 9, no. 1, pp. 3–15, Feb. 2013.

DONGHWA KANG is currently pursuing the bachelor’s degree with the Department of Computer Science and Engineering, Incheon National University (INU), South Korea. His research interests include artificial intelligence, autonomous systems, and real time embedded systems.

INHWA JUNG is currently pursuing the bachelor’s degree with the Department of Computer Science and Engineering, Incheon National University (INU), South Korea. Her research interests include artificial intelligence and autonomous systems.

KILHO LEE received the B.Sc. degree in information and computer engineering from Ajou University, and the M.Sc. and Ph.D. degrees in computer science from KAIST. He is currently an Assistant Professor with the School of AI Convergence, Soongsil University, South Korea. His research interests include system design for real-time embedded systems and cyber-physical systems. He won the Best Paper Award of ACM MobiCom 2019 and IEEE CPSNA, in 2014.

HYEONJOO 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 KAIST, South Korea, in 2012 and 2016, respectively. He is currently 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.

**References**