Real-Time Scheduling for Preventing Information Leakage with Preemption Overheads

Hyeongboo BAEK[1], Jinkyu LEE[1], Jaewoo LEE[2], Pyung KIM[3], Brent Byunghoon KANG[4]
[1]*Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, 16419, Republic of Korea*
[2]*Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA*
[3]*Department of Industrial and Systems Engineering, Seoul National University of Science and Technology, Seoul, 139743, Republic of Korea*
[4]*School of Computing, Korea Institute of Science and Technology, Daejeon, 305338, Republic of Korea*
Corresponding author Brent Byunghoon Kang: brentkang@kaist.ac.kr

***Abstract*—Real-time systems (RTS) are characterized by tasks executing in a timely manner to meet its deadlines as a real-time constraint. Most studies of RTS have focused on these criteria as primary design points. However, recent increases in security threats to various real-time systems have shown that enhanced security support must be included as an important design point, retro-fitting such support to existing systems as necessary. In this paper, we propose a new pre-flush technique referred to as flush task reservation for FP scheduling (FTR-FP) to conditionally sanitize the state of resources shared by real-time tasks by invoking a flush task (FT) in order to mitigate information leakage/corruption of real-time systems. FTR-FP extends existing works exploiting FTs to be applicable more general scheduling algorithms and security model. We also propose modifications to existing real-time scheduling algorithms to implement a pre-flush technique as a security constraint, and analysis technique to verify schedulability of the real-time scheduling. For better analytic capability, our analysis technique provides a count of the precise number of preemptions that a task experiences offline. Our evaluation results demonstrate that our proposed schedulability analysis improves the performance of existing scheduling algorithms in terms of schedulability and preemption cost.**

***Index Terms*—embedded software, real-time systems, scheduling algorithms, security, system analysis and design.**

I. INTRODUCTION

Real-time systems (RTS) control multiple physical devices executing multiple real-time tasks in a timely manner. Since meeting deadlines of real-time tasks is the most important requirement of RTS, most conventional studies have focused on real-time aspects of RTS for several decades while security has not been considered as a first class principle of the design of RTS [1-15]. During this period of time, security has not been accorded sufficient attention at a level commensurate with scheduling algorithm design. However, recent attacks on various RTS have resulted in catastrophizes, including loss of life. We believe that, at this point in time, it is of paramount importance that security be added as a design point for inclusion in any re-design of a RTS. There are many documented attacks on a variety of RTS including telematics units installed in modern automobiles [16-17], industrial control systems [18], and unmanned autonomous vehicles [19-20]. However, applying existing security techniques to RTS cannot be effective because most of these techniques were developed without consideration of the real-time requirements [21-30], and this explains, to a large extent, why RTS are easily compromised.

The lack of support for robust security as an indigenous component of RTS has recently generated a large amount of research. In recent three years, three studies [31-33] have proposed approaches that a pre-flush mechanism to invoke a flush task (FT) is included into real-time scheduling with a design point of real-time systems to eliminate or substantially reduce information leakage that occurs in real-time tasks that share resources and execute with different levels of security.

The initial work [31] first proposed a pre-flush mechanism that will invoke a FT to sanitize the state of all shared resources such as all caches, DRAM and I/O buses. Figure 1 describes an example how a FT is invoked between the executions of two tasks having different security levels; for two tasks A and B, A has a security level higher than B. If B is dispatched immediately subsequent to the dispatch of A, then B is more easily compromised by attackers due to its lower security level and can be used to remove, copy, or corrupt sensitive data that exist on the shared resources in use by A. To prevent this, a FT is invoked at the beginning of the execution of B and executes non-preemptively. The work presented in [31] proposed a new fixed priority (FP) [34-35] real-time scheduling algorithm that includes security constraints regarding conditional invocations of FTs and an associated new response time analysis (RTA) [36-37]. However, this work is limited to non-preemptive FP scheduling and does not provide tight upper-bounds of the worst-case number of FTs thereby illustrating the limited analytic capability for schedulability.

A subsequent work [32] extended the initial work to

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2016R1D1A1B03930580, 2016R1A6A3A11930688) and the Ministry of Science, ICT & Future Planning (NRF-2017R1A2B2002458). This research was also supported by Institute for Information & communication Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 11041244, Development of High Reliable Communications and Security SW for Various Unmanned Vehicles).

Digital Object Identifier 10.4316/AECE.2017.02016

include preemptive FP scheduling and to provide a tighter upper-bound of the worst-case number of FTs for schedulability analysis by transforming the problem of upper-bounding the worst-case number of FTs into a min-cost flow problem [38]. However, this work did not suggest the exact schedulability analysis including the calculation of the exact preemption cost.

Our previous work [33] extended the two works proposed in [31-32] by deriving a new schedulability analysis technique that can compute the exact number of FTs that a task under analysis will experiences during the life of a task. We proposed a new preemptive FP scheduling algorithm referred to as lowest security-level first (LSF). This algorithm employs a new pre-flush technique that invokes FTs in a more intelligent manner, referred as flush task reservation (FTR), which induces the exact schedulability analysis (*i.e.* sufficient and necessary analysis) that can compute the exact number of FTs hindering the execution of a task offline. However, FTR can be used only with LSF scheduling, and its performance is insufficient relative to rate monotonic (RM) [34-35] scheduling, the optimal scheduling algorithm deployed for preemptive FP uniprocessor scheduling.

This paper also addresses the problem of information leakage, the subject of three prior studies, and we propose a new pre-flush mechanism referred to as flush task reservation for FP scheduling (FTR-FP) and a new schedulability analysis technique thereof. FTR-FP is not restricted to a specific FP scheduling algorithm, and will relax the limitation of a FTR that we described above. Unlike the results from three prior conventional [31-33], our new schedulability analysis can estimate not only the exact number of FTs but also the exact number of preemptions that a task will experience. Previous two studies [31], [33] considered introducing a security constraint that a FT is invoked during a context switch from a task with a higher security-level τ_H to a task with a lower security-level τ_L. Invoking a FT during a context switch in both directions, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, can degrade the real-time scheduling performance due to the invocation of more FTs. However, the possibility of information leakage from τ_L to τ_H should not be ignored as it can also compromise the security of the RTS. We show our technique maintain reasonable performance even though we consider both directions of information leakage to better mitigate the possibility of information leakage.

The primary contributions of this paper are the advantages of our proposed techniques compared to the prior techniques in [31-33]:

- By proposing a FT invocation during the context switch in both directions of transition, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, as a security constraint, we enhance the system security relative to the conventional approaches [31], [33] that implement FT invocation at context switch time in a single direction, $\tau_H \rightarrow \tau_L$.
- The new pre-flush technique and schedulability analysis are not limited to a specific FP scheduling, offering an advantage over the previous technique [33] that is restricted to LSF.
- A new schedulability analysis can calculate the precise number of FTs and preemptions, but existing three schedulability analyses [31-33] can upper-bound maximum number of FTs only.

II. Adversary and System Models

In this section, we present the adversary and system models. The design of our adversary and system models are based on production RTS examples in use in various fields as described in [31-33], e.g., an avionics system based on the DO-178B model [32]. Our models also include a new security constraint that extends the security constraints pursuant to [31], [33].

A. Adversary Model

We assume that an adversary understands the nature of the task parameters as well as the RTS scheduling policy. The primary objective of the adversary is to retrieve or otherwise corrupt sensitive data stored in shared resources, e.g., files, databases, upon which real-time tasks are dependent. Tasks executing at a lower security-level have less protection from security threats, so that the probability of successful intrusion initiated by an attacker exceeds the probability of successful intrusion originating from tasks executing at a higher security level. For example, an adversary can launch a side-channel attack by hijacking a task executing at a low security-level to access sensitive data within the shared cache, concurrently accessed by a task executing at a higher security-level.

These attacks against a RTS can be successful in a RTS that supports a multi-tier or hierarchical security level model. An example of such a model is an avionics RTS that implements the DO-178B standard. This model partitions the RTS into sub-systems with defined interfaces. Usually, the development of the sub-systems is contracted to different vendors such that each vendor possesses the necessary security level (clearance) to bid on the sub-system. For example, a sub-system that controls a camera that captures images to be sent to the command center can be designed by a vendor with the required security clearance to use and observe confidential data. On the other hand, the avionics navigation system may be developed by a vendor with less restrictive security credentials as the function of this sub-system is less critical relative to securing shared resources. If the navigation system, developed by a vendor at a low security level, is compromised or corrupts shared sensitive data, then the camera system, created by a vendor with a higher security level, can generate images that can be copied, deleted, or corrupted.

B. System Model

We consider the preemptive FP scheduling of a set of periodic task set τ on a uni-processor [34-35]. A task $\tau_i \in \tau$ is characterized by a 3-tuple (T_i, C_i, D_i), where T_i is the period of a task or minimum inter arrival time, C_i is the worst-case execution time (WCET) and D_i is the relative deadline. We assume implicit deadlines, *i.e.*, D_i is equal to T_i. A task τ_i releases its job j_i periodically according to its period T_i. We use the notation of j_i^n if the job indicates the *n*-th released job of task τ_i. The WCET of a FT is

denoted by C_{ft}. We assume the synchronous release of tasks, so that the first jobs of all tasks are released concurrently, *i.e.*, at the time instant 0. We also assume quantum-based time for real-time scheduling and schedulability analysis.

Previous work [31], [33] considered the security-level of real time tasks and focused on the prevention of information leakage from the resources shared by a task executing at a higher security-level τ_H than a task executing at a lower security-level τ_L. As defined by this security model, a FT should be invoked to sanitize the state of the shared resources only if a lower security-level task τ_L executes immediately after the execution of a higher security-level task τ_H. However, the possibility of information leakage attributed to a context switch from τ_L to τ_H should not be ignored because the change of execution state can also compromise the security of the RTS. Thus, we address the problem of information leakage that arises during both context switches, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, to better reduce the possibility of information leakage. We define a security constraint to be integrated within the real-time scheduling algorithm as follows:

Definition 1: (Security Constraint). Before a job is scheduled, a timing penalty should be spent to cleanse the state of shared resource after another job is executed.

To satisfy the security constraint, a real-time scheduler dispatches a FT when a context switch occurs from job $j_i \rightarrow j_j$ regardless of source and target task security-levels. Thus, our scheduling and schedulability analysis techniques are not sensitive to the security ordering of tasks. We assume that a FT executes concurrently with a job j_i whenever the job is dispatched by the scheduler in accordance with our pre-flush techniques, *i.e.,* the remaining execution time of j_i is increased by C_{ft} to allow adequate time to serialize the task execution followed by the FT execution. Every job must complete its execution within C_{ft} time units prior to its deadline to be schedulable since the proposed scheduling algorithm requires that a FT executes non-preemptively to completion at the preemption or completion of every task.

III. Scheduling Incorporating Security Constraints

In this section, we propose a preemptive real-time scheduling algorithm that includes a new pre-flush technique to satisfy the security constraint described in the previous section. We first provide a detailed discussion of the existing pre-flush algorithms introduced in [31] and [32] and then we present our algorithm as an improvement.

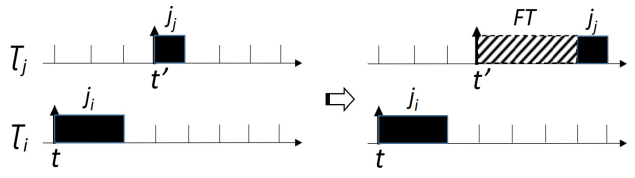


Figure 1. Conventional flush task invocation case

In the existing pre-flush scheme, a FT can be invoked at a time instant at which a job j_i starts its execution after release of the job or resume from any interference caused by higher priority jobs, and shared resource needs to be flushed. The FT executes non-preemptively with highest priority, and this will frequently block the scheduled job j_i, increasing its time-to-completion.

Figure 1 illustrates the effect of interleaving execution of the FT with other jobs competing for the processor, characteristic of an existing pre-flush scheduling algorithm. Considering job j_i that begins or resumes execution at time instant *t*, let us assume that an earliest higher priority job j_j will be released at time instant $t' > t$. As seen in Figure 1, the higher priority job j_j is preempted by the FT (this task has the highest priority) dispatched to prevent information leakage that may have occurred during the context switch, $j_i \rightarrow j_j$. Thus, a higher priority task τ_j can be subject to the execution of a lower priority task τ_i if a FT is invoked between two tasks. Due to this property, it is not straightforward to apply conventional schedulability analysis techniques providing strong analytic capability to this scheduling scheme since most of those are based on the following property:

Definition 2 (Priority Isolation): The execution behavior of higher priority tasks is independent of the execution behavior of lower priority tasks.

For example, standard RTA approach for preemptive FP scheduling is an exact schedulability analysis based on priority isolation only considering interference from higher priority tasks, ignoring the execution behavior of lower priority tasks, to test schedulability of a given task.

While the two documented pre-flush techniques [31-32] violate the principle of priority isolation and the associated schedulability analysis, we propose a new pre-flush technique, which we refer to as flush task reservation for FP scheduling (FTR-FP), accompanied by a new exact schedulability analysis that will conserve the principle of priority isolation. Considering a job j_i that begins or resumes its execution at time instant *t* and an earliest higher priority job j_i released at time $t' > t$, let *e* be the time instant at which j_j will finish its execution. FTR-FP determines an invocation of a FT in every scheduling (release of a job, or resume of a job from any preemption) and invokes FTs in a more intelligent manner creating three cases based on the value of $t' - C_{ft}$:

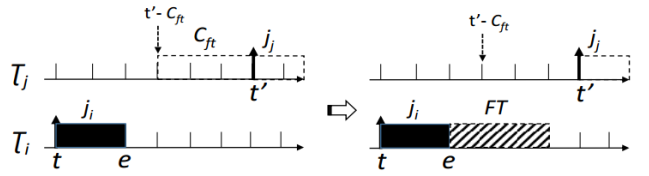
(a) if $e \le t' - C_{ft}$ (the case of Figure 2 (a)), such that C_{ft} is the WCET of a FT, the scheduler reserves a FT to prevent information leakage from j_i to j_j so that a FT will be invoked at the time instant *e*;

(b) in another case for $t \le t' - C_{ft} < e$ (the case of Figure 2 (b)), a FT is reserved and will be invoked at time instant $t' - C_{ft}$; and

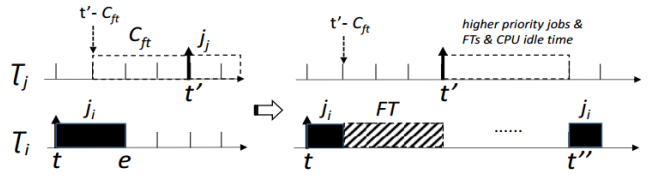
(c) if $t' - C_{ft} < t$ (the case of Figure 2 (c)), the execution of j_i is suspended, and CPU remains idle in the time space between *t* and t' so that a FT cannot interfere τ_j.

We refer to these cases as FTR case (a), FTR case (b) and FTR case (c) respectively. Note that if the sum of remaining

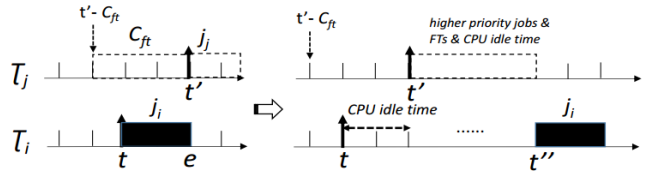
execution time of j_i and C_{ft} is smaller than the remaining time space until the deadline of j_i, then j_i cannot be schedulable. Thus, FTR-FP tests schedulability of j_i in every scheduling, and then it considers three FTR cases for j_i. We assume that the schedulability of j_i in Figure 2 is verified.



(a) FTR case for $e \le t' - C_{ft}$



(b) FTR case for $t \le t' - C_{ft} < e$



(c) FTR case for $t' - C_{ft} < t$

Figure 2. There FTR cases corresponding to the positions of $t' - C_{ft} < t$

In FTR case (a), the time interval between t and t' is sufficiently large for a FT to execute in the time space without interfering not only the job j_i but also a higher priority job j_i. To satisfy the security constraint that we defined, a FT is invoked at time instant e as soon as j_i completes its execution. In FTR case (b), the time space between t and t' is larger than C_{ft}, but it interferes j_j if it is invoked at time instant e, which violates priority isolation. Thus, FTR-FP reserves a FT so that it will invoke at time instant $t' - C_{ft}$ and not interfere j_j but j_i. In FTR case (c), the time space between t and t' is smaller than C_{ft}. Thus, FTR-FP suspends j_i and let CPU idle in the time space between t and t' to prevent j_j from being interfered from a FT; a FT is not invoked in this case to conserve priority isolation. In both FTR cases (b) and (c), j_i is preempted and suspended respectively, and it resumes its execution at time instant t'' after it suffers interference from higher priority jobs including j_j, FTs or CPU idle time; higher priority jobs also can cause CPU idle time due to another higher priority job. FTR-FP iteratively considers three cases for j_i until j_i is in the FTR case (a) so that it completes its execution. If j_i has the highest priority, it executes without any interference, and a FT is invoked as soon as j_i completes its execution. For the job having the highest priority, t' is considered as $+\infty$ since the job does not have higher priority jobs, and it is always in FTR case (a).

FTR-FP is applicable to any FP preemptive scheduling since it invokes a FT on every transition $j_i \rightarrow j_j$, and it does not depend on a specific priority ordering. Figure 2 illustrates that FTR-FP satisfies the security constraint and conserves the principle of priority isolation for two jobs j_i and j_j from tasks τ_i and τ_j respectively. We now show that FTR-FP also satisfies the security constraint and conserves priority isolation for a task set τ containing more than two tasks based on the following reasoning. As j_i begins its execution at time instant t, j_i is the highest priority job in the scheduling queue at time instant t. Also, as an earliest higher priority job j_j releases at t', no jobs having higher priorities than j_i execute in the time space between t and t'. FTR-FP iteratively finds such time spaces at every scheduling of every task. Since a FT is invoked on every transition $j_i \rightarrow j_j$ (FTR cases (a) and (b)) or is not invoked if j_i does not execute (FTR case (c)), FTR-FP always satisfies the security constraint. Also, since j_i and the FT executing with j_i always execute in the time space t and t', priority isolation is conserved; it is guaranteed that j_i and the FT do not interfere higher priority job j_j.

IV. Schedulaibility Analysis

This section presents a new schedulability analysis for our new FP preemptive scheduling employing FTR-FP. We extend previous exact schedulability analysis proposed in [39] aiming at calculating the exact number of preemptions to be applicable our scheduling algorithm. We first introduce the definitions and notations used for our schedulability analysis based on those described in [32], and then, we illustrate our schedulability analysis procedure. We also show that our new schedulability analysis can calculate the precise number of preemptions..

A. Definitions and Notations

As FTR-FP guarantees priority isolation, each task τ_i only needs to consider interference from higher priority tasks including executions, FTs and CPU idle time for its schedulability test but not those from lower priority tasks. As we assume periodic task model, for the schedulability analysis of τ_i, we only need to investigate what happens in the time space from 0 to the least common multiple of periods of tasks including τ_i and higher priority tasks of τ_i. Thus, we define *hyperperiod at level i*, denoted by $H_i = LCM\{T_j\}_{\tau_j \in hep(\tau_i)}$, where $LCM\{T_j\}$ is the least common multiple of T_j and $hep(\tau_i)$ is a set of tasks having a priority higher than or equal to task τ_i. τ_i releases its job σ_i times in *hyperperiod at level i*, where σ_i is calculated by

$$\sigma_i = \frac{H_i}{T_i} = \frac{LCM\{T_j\}_{\tau_j \in hep(\tau_i)}}{T_i} \quad (1)$$

Also, τ_{i-1} releases its job λ_i times in *hyperperiod at level i*, where λ_i is calculated by

$$\lambda_i = \frac{H_i}{T_{i-1}} = \frac{LCM\{T_j\}_{\tau_j \in hep(\tau_i)}}{T_{i-1}} \quad (2)$$

We then define T_i*-mesoid* as an ordered set describing the state of each job j_i of a task τ_i. T_i*-mesoid* consists of three types of time units: a time unit already executed or suspended by CPU idle time, called consumption, a time units for a FT invocation and a time unit still available. We present consumptions by its cardinal inside brackets (*c*), with $c \in \aleph^+$. We use another brackets [*r*], with $r \in \aleph^+$ for WCET of FTs. In addition, we enumerate the sequence of available time units according to the natural numbers. Each of these natural numbers is called availability. Note that the natural numbers of consumptions and FTs present how many availabilities it consumes while that of availability presents its sequence. For example, {(3), [2], 1, 2, (4), [2], 3, 4} is the 15-*mesoid* containing two sets of consumptions having length of 3 and 4 respectively, two FTs where C_{ft} is 2, and four availabilities represented by 1, 2, 3 and 4 respectively. Since each T_i*-mesoid* describes the state of each job, there are σ_i T_i*-mesoid* in H_i, which are the sequences of T_i*-mesoid*. We define $L_i^b = \{M_i^{b,1}, M_i^{b,2}, \cdots, M_i^{b,\sigma_i}\}$ as the sequence of σ_i T_i*-mesoid* before τ_i is scheduled. For example, $L_i^b = \{\{(5), [1], 1, 2, (2), [1]\}, \{1, 2, (3), [1], 3, (3), [1]\}\}$ is a sequence of σ_i = 2 11-*mesoid* where C_{ft} is 1.

We define the cell of L_i^b as a set of consecutive available time units between a FT and consumption. With the definition of cell of T_i*-mesoid*, the above example is rewritten as

$$L_i^b = \{\{(5),[1],[1]_i^1,=\{1,2\},(2),[1]\}, \{[1]_i^2=\{1,2\},(3),[1],[2]_i^2=\{3\},(3),[1]\}\} \quad (3)$$

where $[m]_i^j$ denotes the m^{th} cell in j^{th} T_i*-mesoid* of L_i^b. The number of time units in the cell $[m]_i^j$ is denoted by $|[m]_i^j|$.

As we can see from the example, the WCET of the first job j_i^1 and the second job j_i^2 should not exceed 1; otherwise task τ_i cannot be schedulable. We call $L_i^a = \{M_i^{a,1}, M_i^{a,2}, \cdots, M_i^{a,\sigma_i}\}$ the sequence of σ_i T_i*-mesoid* of task τ_i after τ_i is scheduled. L_i^a is resulted from L_i^b as the available time units will have been consumed up by executions of j_i and FTs associated it to the response time within each mesoid $M_i^{b,k}$, k = 1, ⋯, σ_i of task τ_i after τ_i is scheduled. The process of building L_i^a from L_i^b, and L_i^{b+1} from L_i^a respectively will be detailed in the next section. To sum up, for every task τ_i, we have

$$\tau_i : \begin{cases} L_i^b = \{M_i^{b,1}, M_i^{b,2}, \cdots, M_i^{b,\sigma_i}\} \\ L_i^a = \{M_i^{a,1}, M_i^{a,2}, \cdots, M_i^{a,\sigma_i}\} \end{cases} \quad (4)$$

B. Schedulability Analysis

Our new schedulabililty analysis approach for FP preemptive scheduling employing FTR-FP tests scheduability of each task in a top-down manner from the highest priority task to the lowest priority task. As FTR-FP conserves priority isolation, our schedulability analysis only needs to consider interference from higher priority tasks. The analysis conducts two phases for each task: inheritance phase and FTR phase.

- In inheritance phase, L_i^b of τ_i inherits L_{i-1}^a λ_i times so that all availabilities in L_i^b are filled with the time units of L_{i-1}^a in a cyclic manner. In detail, $(x^*|L_{i-1}^a|+y)^{th}$ time unit in L_i^b is filled with y^{th} time unit in L_{i-1}^a, where $|L_{i-1}^a|$ is the number of time units in L_{i-1}^a, $0 \le x < \lambda_i$, and $1 \le y < |L_{i-1}^a|$.
- In FTR phase, availabilities in L_i^b are consumed by executions of jobs of τ_i, CPU idle time, and FTs reserved according to the policy of FTR-FP investigating which case each job of τ_i is in among three FTR cases, and then L_i^a is constructed for L_{i+1}^a of τ_{i+1} if any.

Before we generalize our schedulability analysis process to support for *n* tasks, we describe the procedure of schedulaility analysis for two tasks, τ_1 and τ_2, assuming that τ_1 has higher priority than τ_2. We further assume that τ_1 and τ_2 have task parameters (8, 1, 8) and (12, 3, 12) respectively, and the WCET of a FT C_{ft} is 2. As τ_1 has the highest priority, schedulability analysis does not conduct inheritance phase for τ_1 since it does not have higher priority tasks. In FTR phase, τ_1 produces C_1 consumptions, a FT and $T_1 - (C_1 + C_{ft})$ availabilities in each T_i*-mesoid* since each job j_1 of τ_1 can consume any availability in T_1 and never be preempted. Thus, T_1*-mesoid* of τ_1 are described as

$$\tau_1 : \begin{cases} L_1^b = \{M_1^{b,1}\} = \{\{1,2,3,4,5,6,7,8\}\} \\ L_1^a = \{M_1^{a,1}\} = \{\{(1),[2],1,2,3,4,5\}\} \end{cases} \quad (5)$$

Then, the inheritance phase and the FTR phase are conducted for task τ_2. As T_1 and T_2 are 8 and 12 respectively, $H_2 = LCM\{8, 12\}$ is 24. In inheritance phase of τ_2, as σ_2 = 2 and λ_2 = 3, we construct L_2^b composed of a sequence of two 12-mesoids by inheriting time units of L_1^a three times as described above. Based on the mechanism, we have

$$L_2^a = \{M_2^{b,1}, M_2^{b,2}\} = \{\{(1),[2],1,2,3,4,5,(1),[2],6\}, \{1,2,3,4,(1),[2],5,6,7,8,9\}\} \quad (6)$$

With the definition of cell, L_2^b can also be written as

$$L_2^a = \{\{(1),[2],[1]_2^1=\{1,2,3,4,5\},(1),[2],[2]_2^1=\{6\}\}, \{[1]_2^2=\{1,2,3,4\},(1),[2],[2]_2^2=\{5,6,7,8,9\}\}\} \quad (7)$$

We then perform FTR phase satisfying the security constraint and priority isolation. In FTR phase, we investigate each cell sequentially and see which case each job starting or resuming its execution is in among three FTR cases. According to the policy of FTR-FP for each case, time units of each cell are consumed by execution of each job and FTs. In detail, FTR phase is conducted by following policies:

- A single job is released in a T_i-*mesoid*;
- Consumptions of a job begin with the first availability in a cell;
- Availabilities are consumed by executions of j_i, FTs and CPU idle time by the policy of corresponding case that each job is in among three FTR cases until the job completes its execution.

Subsequent to the FTR phase of our schedulability analysis, L_2^b of τ_2 becomes

$$L_2^a = \{\{(1),[2],\underline{(3),[2]},(1),[2],1\}, \{\underline{(2),[2]},(1),[2],\underline{(1),[2]},1,2\}\} \quad (8)$$

New consumptions of jobs of τ_2 and consumed time units by FTs are underlined. The first job of L_2^b consumes three availabilities from the first availability to the third availability in $M_2^{b,1}$, and then a FT follows as the job is in the FTR case (a). The second job of L_2^b is in the FTR case (b) first resulting in two consumptions and consumed time units by a FT, and then it is preempted. After preemption from a single consumption and a FT, it resumes its execution and it completes its execution producing a single consumption and a FT, which is in the FTR case (a).

After the completion of both phases, the schedulability of the task τ_2 is verified as schedulabilities of all jobs in the hyper period H_2 are tested. In the above example, τ_2 is schedulable since all jobs in H_2 complete its execution within its deadline D_2.

Our schedulability analysis also can calculate the exact number of preemptions by counting preemption whenever a job j_i does not complete its execution in a cell; it is in the FTR case (b). For example, the second job in L_2^b experiences a preemption by a FT following the job at the third time unit in $M_2^{b,2}$, which is in the FTR case (b). Assuming that the cost of preemption denoted by C_p is 1, the resulted L_2^b is rewritten as

$$L_2^b = \{\{(1),[2],\underline{(3),[2]},(1),[2],1\}, \{\underline{(2),[2]},(1),[2],\underline{(2),[2]},1\}\} \quad (9)$$

If the cost of preemption exceeds 2, the second job cannot be schedulable.

```
1: for i = 2 to n do
2:     σ_i ← (H_i − LCM(T_j)_{τ_j ∈ hep(τ_i)}) / T_i
3:     for x = 1 to σ_i do
4:         Build empty M_i^{b,x} composing L_i^b
5:     end for
6:     INHERITANCE-PHASE (L_{i−1}^a, L_i^b)
7:     Build all cells in L_i^b
8:     if FTR-PHASE (L_i^b, σ_i) = fail then
9:         return unschedulable
10:    end if
11:    Build L_i^a by duplicating L_i^b
12: end for
13: return schedulable
```

Figure 3. Algorithm for Schedulability Analysis

```
1: for x = 0 to λ_i-1 do
2:     for y = 1 to |L_{i−1}^a| do
3:         ((x ∗ |L_{i−1}^a| + y)^th time unit in L_i^b) ← (y^th time unit in L_{i−1}^a)
4:     end for
5: end for
```

Figure 4. Algorithm for Inheritance phase(L_{i-1}^a , L_i^b)

```
1: for x = 1 to σ_i do
2:     y ← C_i
3:     for m = 1 to the number of cells in M_i^{b,x} do
4:         if y + C_ft ≤ |[m]_i^x| then
5:             Produce y consumptions and a FT
6:             y ← 0
7:             break
8:         else if C_ft ≤ |[m]_i^x| < y + C_ft then
9:             Produce |[m]_i^x| − C_ft consumptions and a FT
10:            y ← y − (|[m]_i^x| − C_ft)
11:            y ← y + C_p
12:        else
13:            Produce |[m]_i^x| consumptions
14:        end if
15:    end for
16:    if y > 0 then
17:        return fail
18:    end if
19: end for
20: return success
```

Figure 5. Algorithm for FTR phase(L_i^b , σ_i)

Figure 3 presents the procedure of our schedulability analysis using functions of inheritance phase (Figure 4) and FTR phase (Figure 5) for n tasks in detail. Since both our schedulability analysis and existing one proposed in [33] are extended from the top-down based exact schedulability analysis introduced in [39], to consider a FT invocation, two approaches share the similar framework; they investigate what happens in the interval of *hyperperiod at level i* and inherits the schedules of higher priority tasks since they target FP scheduling in inheritance phase (Figures 3 and 4). However, our approach has the following two main advanced parts from the existing one: we support different flush task reservation mechanism preventing information leakages in both directions $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$ while the existing one is only applicable to $\tau_H \rightarrow \tau_L$ only, and we can calculate the preemption cost exactly by counting the number of preemption happened exactly, which are implemented in FTR phase (Figures 5).

Since the highest priority task τ_1 is never preempted, the loop starts from the index of the second priority task τ_2 as we conduct analysis towards lower priority task. The schedulability analysis for τ_i has following steps:

(1) It first computes *hyperperiod at level i* and the number of instance within hyperperiod (Line 2 in Figure 3);
(2) Then it builds T_i *-mesoid* where all time units are available composing L_i^b (Lines 3-5 in Figure 3);
(3) The sequence L_i^b is derived by inheriting L_{i-1}^b λ_i times in inheritance phase (Line 6 in Figure 3);
(4) It builds all cells based on the availabilities in L_i^b (Line 7 in Figure 3);
(5) Then, availabilities in cells in L_i^b are filled with consumptions and FTs according to the policies of FTR cases (Lines 8-9 in Figure 3). In this step, schedulability of a job is tested;
(6) It constructs L_i^a by duplicating L_i^b for the next task if any (Line 10 in Figure 3); and
(7) If all tasks satisfy the schedulability condition, then it concludes that the task set is schedulable (Line 13 in Figure 3).

The detail procedures of inheritance phase and FTR phase are presented in Figure 4 and 5 respectively. In inheritance phase, the time units in L_i^b are filled with the time units in L_{i-1}^b in a cyclic manner as we described previously (Lines 1-5 in Figure 4). In FTR phase, one of the policies of three FTR cases is applied as many times as the number of cells in $M_i^{b,x}$, for a job in L_i^b; FTR cases (a), (b) and (c) are corresponding to Lines 4-7, 8-11 and 12-14 in Figure 5 respectively. Note that preemption cost is added only in FTR case (b) where preemption occurs (Line 11 in Figure 5). If any job does not complete its execution even when all cells are consumed by executions, CPU idle time or FTs, then the job is unschedulable (Lines 16-17 in Figure 5) otherwise it is schedulable (Line 20 in Figure 5).

V. Evaluation

In this section, we evaluate our analysis technique compared to the conventional approaches. We first illustrate the simulation environment of our evaluation, and then we discuss the simulation results of considered techniques. For the main metric of performance, we use the schedulability ratio defined as the number of task sets deemed schedulable by an analysis to the total number of generated task sets.

A. Simulation environment

We randomly generate task sets implemented by our own JAVA program, which are based on the task sets generation method used in [31-33]. Then, we investigate how many tasks are deemed schedulable by each considered schedulability analysis technique (also implemented by JAVA program). Note that as our interest is the schedulability ratio of each considered technique, the evaluation results are independent on our computing capability of simulation environment, e.g., CPU or memory; it only depends on the mathematical calculation of each considered schedulability analysis technique. The base utilization groups $[0.02 + 0.1 \cdot i, 0.08 + 0.1 \cdot i]$ for $i = 0, \ldots, 9$, are generated, and each group has 200 task sets. For example, the first group of base utilization has a range from 0.02 to 0.08, and the last group has a range from 0.92 to 0.98. Then, each base utilization group contains task sets whose total sum of task utilizations (i.e., sum of C_i / T_i of tasks in each task set) belonging to its range of base utilization.

Table I. Parameters for Experiments

Parameter	Value
Number of task, N	$N \in \{3, 4, \cdots, 10\}$
Task period, T_i	$T_i \in \{50, 100, \cdots, 1000\}$
Task execution time, C_i	$C_i \in \{5, 6, \cdots, 50\}$
FT overhead, C_{ft}	$C_{ft} \in \{0, 1, \cdots, 10\}$

Table I describes the task parameters used for our evaluation. Each task set can contain at least 3 tasks and at most 10 tasks. Each task has a period $T_i \in \{50, 100, \cdots, 1000\}$ and an execution time $C_i \in \{5, 6, \cdots, 50\}$. The deadline of each task D_i is equal to its period. The values are based on actual resources in actual architecture (e.g. the core i7 or Tegra 3 device) as described in [33]. We generate 2,000 task sets for each value of C_{ft}, and therefore, 20,000 tasks sets are generated in total.

B. Simulation results

For our evaluation, we consider RM for scheduling algorithm and RTA for schedulability analysis as techniques for comparison since those are the de facto standards techniques, which are the optimal preemptive FP scheduling and exact schedulability analysis on uni-processor. We also consider LSF schedulability analysis proposed in [33] for comparison to FTR-FP since, likewise our mechanism, it is also extends the schedulability analysis suggested in [39], and it supports LSF, one of the preemptive scheduling algorithms. Following five schedulability analyses designed for FP preemptive scheduling algorithm on uniprocessor are considered:

- RTA approach for preemptive RM scheduling algorithm not considering a security constraint (RM−RTA),
- RTA approach with the obviously bounded number of FT invocations (the number of FT invocation is $2 * N_{hep(i)} + 1$, where $N_{hep(i)}$ is the number of higher or equal priority jobs for each task τ_i) for preemptive RM scheduling, proposed in [31-32] (RM−RTA−ob),
- RTA approach for preemptive LSF scheduling algorithm not considering a security constraint (LSF-RTA),
- LSF schedulability analysis exactly counting the maximum number of FT invocations proposed in [33] (LSF−FTR), and
- the schedulability analysis introduced in Section 4 (RM-FTR−FP).

Note that RM-RTA and LSF-RTA do not consider a security constraint and the rest techniques consider the security constraint. RM-RTA-ob upper-bounds the maximum number of FTs, and LSF-FTR and RM-FTR-FP exactly count that. The performances of all techniques except RM-RTA-ob are corresponding to those of scheduling algorithms respectively since the techniques are the exact schedulability analyses providing exact predictability.

We first investigate how performances of three RM

schedulability analyses change according to the varying value of FT overhead. Figure 6 shows the changing schedulable ratio of RM-FTR-FP compared to those of RM-RTA and RM-RTA-ob with the varying value of overheads of FT ranging from 0 to 10. As shown in Figure 6, RM-RTA makes most task sets schedulable, and it is not affected by the varying value of FT overhead since it does not consider a security constraint. On the other hands, the schedulable ratio of RM-RTA-ob drops sharply due to its pessimistic bound on the maximum number of FT; it is below 50% for an FT overhead of 10. In the case of RM-FTR-FP that is the proposed technique in this paper, it maintains a high schedulable ratio in spite of the increasing value of FT overhead as it can calculates the exact number of FT; the ratio is approximately 79% even with an FT overhead of 10. Thus, Figure 6 demonstrates that our proposed technique can enhance security while reducing performance degradation.

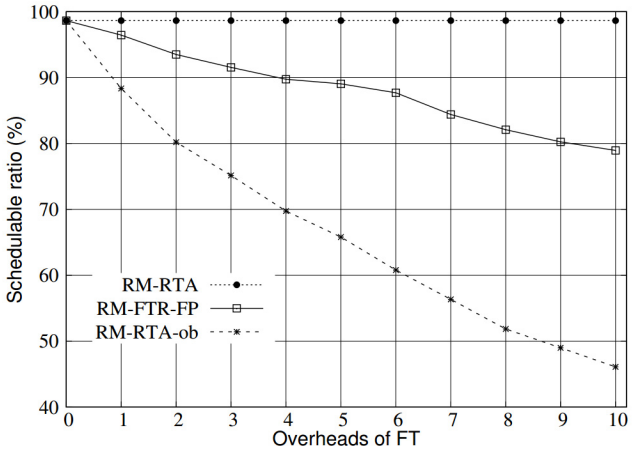


Figure 6. Schedulability ratio of three RM schedulability analyses according to varying value of FT overhead

In our second simulation, we compare our technique (*i.e.* RM-FTR-FP) with two existing LSF schedulability analyses (*i.e.* LSF-RTA and LSF-FTR). As both LSF-FTR and RM-FTR-FP are based on the schedulability analysis technique proposed in [39], initially designed to calculate the exact number of preemptions, both mechanisms are also capable to calculate the exact number of preemptions. However, due to the performance gap between LSF and RM scheduling algorithms, LSF-FTR shows limited analytic capability as LSF-FTR is for LSF scheduling algorithm whose performance is inferior to the performance of the RM scheduling algorithm. Figure 7 compares the schedulability ratio of three schedulability analyses: RM-FTR-FP, LSF-RTA and LSF-FTR. As show in Figure 7, LSF-RTA shows constant performance in spite of varying values of FT overhead as it does not consider a security constraint. Even though RM-FTR-FP considers a security constraint, it outperforms LSF-RTA and LSF-FTR as it is based on RM scheduling algorithm.

As shown in Figure 6 and 7 there is about 23% gap of schedulable ratio between RM-RTA and LSF-RTA. Although LSF-FTR shows similar performance degradation rate to RM-FTR-FP for increasing value of FT overhead, its schedulable ratio is limited to below that of LSF-RTA (about 76%). Therefore, as we made FTR to be applicable to any FP scheduling algorithm including RM that is the de facto standard FP scheduling algorithm in uni-processor, we significantly improved the performance of FTR compared to LSF-FTR; we improved the scheduability ratio by 23% for $C_{ft} = 0$ and 11% for $C_{ft} = 10$ respectively.

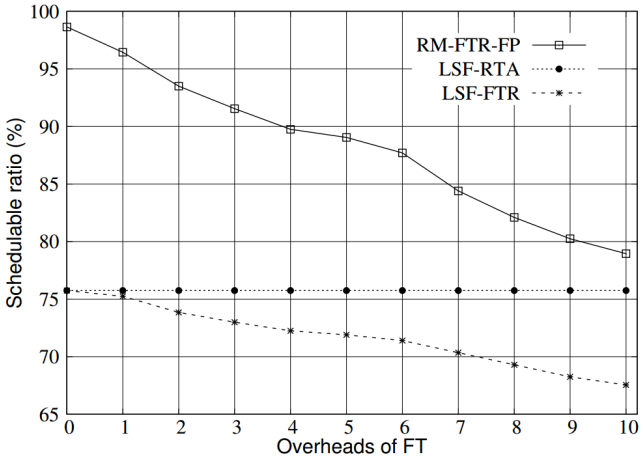


Figure 7. Schedulable ratio of proposed schedulability analysis compared to two LSF schedulability analyses

As one of the major advantage of our schedulability analysis compared to the existing three techniques in [31-33] is a capability to exactly calculate the number of preemption that occurs in the scheduling of preemptive scheduling algorithm. Note that although the schedulability analysis technique proposed in [33] has the similar framework of ours that is advantageous to calculate the number of preemptions, the method to calculate the preemption was not implemented in [33]. In following evaluation, we measure the average actual number of preemptions incurred by each task sets during 100,000 time units. Since existing three techniques are not capable to calculate the exact number of preemptions, we calculate the number of jobs actually released during scheduling as the upper-bound of the number of preemptions since the number of preemption is no larger than the number of jobs actually released in preemptive scheduling. As we considered RM scheduling algorithm for the comparison with techniques in [31-32] and LSF scheduling algorithm for the comparison with the technique in [33], in the previous two evaluation, we measures the number of jobs actually released and preemptions actually incurred during scheduling of RM and LSF. Table II present the average actual number of jobs released and preemptions incurred during 100,000 time units of the scheduling of RM and LSF. Note that two scheduling algorithms are based on the same set of tasks having the same take parameters, and therefore the average actual numbers of jobs actually released thereof during 100,000 time units are the same, e.g., 1234.8. As shown in Table II, two scheduling algorithm incurs similar number of preemptions; each average actual number of preemptions incurred during RM and LSF is 542.8 and 579.3 respectively. The ratio of jobs to preemptions of RM and LSF are 227.4% and 213.1% respectively. This indicates that our schedulability analysis can be performed much better than the existing three techniques when the cost of preemption increases.

TABLE II. AVERAGE ACTUAL NUMBER OF JOBS AND PREEMPTIONS RELEASED AND INCURRED BY EACH TASK SETS DURING 100,000 TIME UNITS

	jobs	**preemptions**	**Jobs/preemptions**
RM	1234.8	542.8	227.4%
LSF	1234.8	579.3	213.1%

VI. Related Work

Previous studies dealt with the security problem by combining the real-time requirement and security mechanisms. Xie and Lin considered periodic task scheduling requiring security service which has the varying overhead in relation to the level of service [40-41]. They extended the conventional scheduler to satisfy the real-time requirement and proposed new scheduler while maximizing the level of security service. On the other hand, we focused on a more concreate security problem, i.e., information leakage on shared resources and security solution, i.e., FT, and aimed at improving schedulability of FP scheduling while meeting security constraint that we defined in this paper.

Some works have addressed the information leakage problem within real-time database systems with embedded security constraints [42-44]. Son [43] proposed transaction scheduling and a concurrency control algorithm satisfying security and real-time requirements. This work included definitions of metrics that will measure the extent to which the security requirements and the concurrency control algorithm are satisfied, allowing a trade-off between the security and real-time requirements [42]. Son [44] also proposed a concept of partial security to enable a trade-off between the two functions cited above. While these studies focused on the area of real-time database system, we aimed at developing new real-time scheduling algorithm and schedulability analysis thereof.

There have been works that suggested architectural frameworks to solve problems of intrusion detection in real-time or cyber physical systems [45-48]. The key idea is to develop hardware/software mechanisms using characteristics of real-time or cyber physical systems to protect against security vulnerabilities. While these studies aimed at detecting of intrusion in real-time systems, we focused on preventing information leakage on the shared resources.

As a FT is invoked on the transition between executions of two jobs, the number of FT invocations interfering with a job under analysis is closely related to the number of preemptions occurring during the execution of a job. An existing work suggested a new schedulability analysis that can exactly count the number of preemptions for preemptive FP scheduling algorithms [39]. Thus, the previous work provided many hints regarding the computation of the precise number of FTs in schedulability analysis. However, the approach in [39] did not consider security constraint.

The recent three works in [31-33] are most closely related to our study, which aim at preventing information leakage by conditionally cleansing the state of shared resource in real-time systems. The initial work [31] first proposed the pre-flush mechanism to invoke a FT, the following work [32] extended it to be applicable more general scheduling model, the latest one [33] proposed exact schedulability analysis that can calculate the exact number of FTs. However those studies are not capable to calculate the exact number of preemptions, and show limited analytical performance or applicability to a single direction of information leakage.

VII. Conclusion

In this paper, we defined a new security constraint to mitigate the possibility of information leakage arisen on real-time tasks sharing resources in real-time systems. We then proposed a new pre-flush mechanism referred to as FTR-FP relaxing the limitation of FTR [33] that its schedulability analysis is limited to LSF. We show that our techniques are applicable to all FP scheduling including RM which is the de facto scheduling algorithm on uni-processor. We also showed that FTR-FP invokes FT during the context switch in both directions of transition, $\tau_H \rightarrow \tau_L$ and $\tau_L \rightarrow \tau_H$, as it meet security constraint that we defined, FR-FP enhances the system security relative to the conventional approaches [31], [33] that implement FT invocation at context switch time in a single direction, $\tau_H \rightarrow \tau_L$.

Moreover, our schedulability analysis can calculate not only the exact number FTs hindering the execution of a job but also the exact number of preemptions while existing three studies [31-33] focused on bounding the maximum number of FTs only. Our experimental results showed that our mechanism maintains reasonable performance in terms of schedulability and preemption cost.

For the future works, we would like to extend our approach into multi-processor platform. We are also planning to implement our scheduling algorithm incorporating FTR-FP and schedulability analysis techniques into real-time operating system and demonstrate how our approach is working well practically.

REFERENCES

[1] A. Biondi, G. Buttazzo, M. Bertogna, "Schedulability analysis of hierarchical real-time systems under shared resources," IEEE Transactions on Computers, vol. 65, issue. 5, pp. 1593 – 1605, 2016. doi:10.1109/TC.2015.2444833.

[2] A. Melani, et al., "Exact response time analysis for fixed priority memory-processor co-scheduling," IEEE Transactions on Computers, vol. PP, issue. 99, pp. 1 – 110, 2016. doi:1109/TC.2016.2614819.

[3] A. Melani, M. Bertogna, V. Bonifaci, A. M. Spaccamela, G. Buttazzo, "Schedulability analysis of conditional parallel task graphs in multicore systems," IEEE Transactions on Computers, vol. 66, issue. 2, pp. 339 – 353, 2017. doi:10.1109/TC.2016.2584064.

[4] X. Hua, C. Guo, H. Wu, D. Lautner, S. Ren, "Analysis for real-time task set on resource with performance degradation and dual-level periodic rejuvenations," IEEE Transactions on Computers, vol. 66, issue. 3, pp. 553 – 559, 2017. doi:10.1109/TC.2016.2602833.

[5] M. Bambagini, M. Marinoni, H. Aydin, G. Buttazzo, "Energy-aware scheduling for real-time systems: a survey," ACM Transactions on Embedded Computing Systems, vol. 15, no. 7, issue. 1, pp. 1 – 33, 2016. doi:10.1145/2808231.

[6] M. Haque, H. Aydin, D. Zhu, "On reliability management of energy-aware real-time systems through task replication," IEEE Transactions on Parallel and Distributed Systems, vol. 28, issue. 3, pp. 813 – 825, 2017. doi:10.1109/TPDS.2016.2600595.

[7] S. Mittal, "A survey of techniques for improving energy efficiency in embedded computing systems," International Journal of Computer Aided Engineering and Technology, vol. 6, issue. 4, pp. 1 – 12, 2014. doi: 10.1504/IJCAET.2014.065419.

[8] C. Krishna, "Fault-tolerant scheduling in homogeneous real-time systems," ACM Computing Surveys, vol. 46, issue. 4, no. 48, pp. 1 – 48, 2014. doi:10.1145/2534028.

[9] H. M. Mora, D. Gil, J. F. C. López, M. T. S. Pont, "Flexible framework for real-time embedded systems based on mobile cloud computing paradigm," Mobile Information Systems, vol. 2015, id. 652462, pp. 1 – 14, 2015. doi: 10.1155/2015/652462.

[10] A. Saifullah, et al., "Parallel real-time scheduling of DAGs," IEEE Transactions on Parallel and Distributed Systems, vol. 25, issue. 12, pp. 3242 – 3252, 2014. doi:10.1109/TPDS.2013.2297919.

[11] J. Leung, J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," Performance Evaluation, vol. 2, pp. 237 – 250, 1982. doi:10.1016/0166-5316(82)90024-4.

[12] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," IEEE Real-Time Systems Symposium (RTSS), 2007, pp. 239 – 243. doi:10.1109/RTSS.2007.35.

[13] H. Chwa, et al., "Extending Task-level to Job-level Fixed Priority Assignment and Schedulability Analysis Using Pseudo-deadlines," IEEE Real-Time Systems Symposium (RTSS), pp. 51 – 62, 2012. doi:10.1109/RTSS.2012.58.

[14] N. C. Audsley, "On priority assignment in fixed priority scheduling," Information Processing Letters, vol. 79-1, pp. 39 – 44, 2001. doi: 10.1016/S0020-0190(00)00165-4

[15] N. Guan, M. Stigge, W. Yi, Ge Yu, "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling," IEEE Real-Time Systems Symposium (RTSS), pp. 51 – 62, 2009. doi:10.1109/RTSS.2009.11

[16] K. Koscher, et al., "Experimental security analysis of a modern automobile," IEEE Symposium on Security and Privacy (SP), pp. 447 – 462, 2010. doi:10.1109/SP.2010.34.

[17] K. Koscher, A. Czeskis, F. Roesner, "Experimental Security Analysis of a Modern Automobile," IEEE Symposium on Security and Privacy (SP), pp. 447 – 462, 2010. doi: 10.1109/SP.2010.34.

[18] K. Fisher, " Using formal methods to enable more secure vehicles: DARPA's HACMS program," ACM SIGPLAN international conference on Functional programming, pp. 1 – 1, 2014. doi:10.1145/2628136.2628165.

[19] J. Pleban, R. Band, R. Creutzburg, "Hacking and securing the AR.Drone 2.0 quadcopter: Investigations for improving the security of a toy," SPIE - The International Society for Optical Engineering, pp. 1 – 12, 2014. doi:10.1117/12.2044868.

[20] J. Son, J. Alves-Foss, "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems," IEEE Information Assurance Workshop, pp. 361 – 368, 2006. doi: 10.1109/IAW.2006.1652117.

[21] J. Li, et al., "Analysis of federated and global scheduling for parallel real-time tasks," Euromicro Conference on Real-Time Systems (ECRTS), pp. 85 – 96, 2014. doi:10.1109/ECRTS.2014.23.

[22] R. Pathan, "Real-time scheduling algorithm for safety-critical systems on faulty multicore environments," Real-Time Systems, vol. 53, issue. 1, pp 45 – 81, 2017. doi:10.1007/s11241-016-9258-z.

[23] A. Melani, R. Mancuso, D. Cullina, M. Caccamo, L. Thiele, "Optimizing resource speed for two-stage real-time tasks," Real-Time Systems, vol. 53, issue. 1, pp 82 – 120, 2017. doi: 10.1007/s11241-016-9259-y.

[24] J. Goossens, E. Grolleau, L. Grosjean, "Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms," Real-Time Systems, vol. 52, issue. 6, pp. 808 – 832, 2016. doi:10.1007/s11241-016-9256-1.

[25] J. Chen, "Federated scheduling admits no constant speedup factors for constrained-deadline DAG task systems," Real-Time Systems, vol. 52, issue. 6, pp. 833 – 838, 2016. doi:10.1007/s11241-016-9255-2.

[26] E. Massa, G. Lima, P. Regnier, G. Levin, S. Brandt, "Quasi-partitioned scheduling: optimality and adaptation in multiprocessor real-time systems," Real-Time Systems, vol. 52, issue 5, pp. 566 – 597, 2016. doi:10.1007/s11241-016-9251-6.

[27] S. Altmeyer, R. Douma, W. Lunniss, R. Davis, "On the effectiveness of cache partitioning in hard real-time systems," Real-Time Systems, vol. 52, issue. 5, pp. 598 – 643, 2016. doi:10.1007/s11241-015-9246-8.

[28] M. Xu, et al., "Cache-aware compositional analysis of real-time multicore virtualization platforms," Real-Time Systems, vol. 51, issue. 6, pp. 675 – 723, 2015. doi:10.1007/s11241-015-9223-2.

[29] H. Zeng, M. Natale, "Computing periodic request functions to speed-up the analysis of non-cyclic task models," Real-Time Systems, vol. 51, issue 4, pp. 360 – 394, 2015. doi:10.1007/s11241-014-9209-5.

[30] Jing Li, et al., "Global EDF scheduling for parallel real-time tasks," Real-Time Systems, vol. 51, issue 4, pp. 395 – 439, 2015. doi:10.1007/s11241-014-9213-9.

[31] S. Mohan, M. Yoon, R. Pellizzoni, R. Bobba, "Real-time systems security through scheduler constraints," Euromicro Conference on Real-Time Systems (ECRTS), pp. 129 – 140, 2014. doi:10.1109/ECRTS.2014.28.

[32] R. Pellizzoni, et al., "A generalized model for preventing information leakage in hard real-time systems," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 271 – 282, 2015. doi:10.1109/RTAS.2015.7108450.

[33] H. Baek, J. Lee, Y. Lee, H. Yoon, "Preemptive real-time scheduling incorporating security constraint for cyber physical systems," IEICE Transactions on Information and Systems, vol. E99-D, no. 8, pp. 2121 – 2130, 2016. doi:10.1587/transinf.2015EDP7493.

[34] C. Liu, J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of ACM, vol. 20 – 1, pp. 46–61, 1973. doi:10.1145/321738.321743.

[35] M. Stigge, W. Yi, "Combinatorial abstraction refinement for feasibility analysis of static priorities," Real-Time Systems, vol. 51, issue. 6, pp. 639 – 674, 2015. doi:10.1007/s11241-015-9220-5.

[36] M. Jeseph, P. Pandya, "Finding response times in a real-time system," BCS Computer Journal, vol. 29-55, pp. 390 – 395, 1986. doi:10.1093/comjnl/29.5.390.

[37] W. Kang, J. Chung, "Energy-efficient response time management for embedded databases," Real-Time Systems, vol. 53, issue. 2, pp. 228 – 253, 2017. doi:10.1007/s11241-016-9264-1.

[38] A. Erlebach, "Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow," Jornal of Scheduling, vol. 7, pp. 233–241, 2004. doi:10.1023/B:JOSH.0000019682.75022.96.

[39] P. Yomsi, Y. Sorel, "Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems," Euromicro Conference on Real-Time Systems (ECRTS), pp. 280 – 290, 2007. doi:10.1109/ECRTS.2007.15.

[40] T. Xie, X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," ACM Transactions on Embedded Computing Systems, vol. 6-3, 2007. doi:10.1145/1275986.1275992.

[41] M. Lin, et al., "Static security optimization for real-time systems," IEEE Transactions on Industrial Informatics, vol. 5-1, pp. 22 – 37, 2009. doi:10.1109/TII.2009.2014055.

[42] Q. Ahmed, S. Vrbsky, "Maintaining security in firm real-time database systems," Conference on Computer Security Applications, pp. 83 – 90, 1998. doi:10.1016/S0164-1212(01)00111-X.

[43] S. Son, "Supporting timeliness and security in real-time database systems," Euromicro Workshop on Real-Time Systems, 1997, pp. 266 – 273. doi:10.1109/EMWRTS.1997.613794.

[44] S. Son, C. Chaney, N. Thomlinson, "Partial security policies to support timeliness in secure real-time databases," IEEE Symposium on Security and Privacy (SP), pp. 136 – 147, 1998. doi:10.1109/SECPRI.1998.674830.

[45] S. Mohan, et al., "S3a: secure system simplex architecture for enhanced security and robustness of cyber physical systems," ACM international conference on High confidence networked systems, pp. 65 – 74, 2013. doi:10.1145/2461446.2461456.

[46] G. Suh, J. Lee, D. Zhang, S. Devadas, "Secure program execution via dynamic information flow tracking," International conference on Architectural support for programming languages and operating systems, pp. 85 – 96, 2004. doi:10.1145/1024393.1024404.

[47] M. Yoon, S. Mohan, J. Choi, J. Kim, L. Sha, "Securecore: A multicore based intrusion detection architecture for real-time embedded systems view document," Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 21 – 31, 2013. doi:10.1109/RTAS.2013.6531076.

[48] C. Zimmer, B. Bhat, F. Mueller, S. Mohan, "Time-based intrusion detection in cyber-physical systems," ACM/IEEE International Conference on Cyber-Physical Systems, pp. 109 – 118, 2010. doi:10.1145/1795194.1795210.