PAPER

Preemptive Real-Time Scheduling Incorporating Security Constraint for Cyber Physical Systems

Hyeongboo BAEK[†], ***Student Member***, **Jaewoo LEE**[††a)], **Yongjae LEE**[†], ***and*** **Hyunsoo YOON**[†], ***Nonmembers***

SUMMARY Since many cyber-physical systems (CPSs) manipulate security-sensitive data, enhancing the quality of security in a CPS is a critical and challenging issue in CPS design. Although there has been a large body of research on securing general purpose PCs, directly applying such techniques to a CPS can compromise the real-time property of CPSs since the timely execution of tasks in a CPS typically relies on real-time scheduling. Recognizing this property, previous works have proposed approaches to add a security constraint to the real-time properties to cope with the information leakage problem that can arise between real-time tasks with different security levels. However, conventional works have mainly focused on non-preemptive scheduling and have suggested a very naive approach for preemptive scheduling, which shows limited analytical capability. In this paper, we present a new preemptive fixed-priority scheduling algorithm incorporating a security constraint, called lowest security-level first (LSF) and its strong schedulability analysis to reduce the potential of information leakage. Our simulation results show that LSF schedulability analysis outperforms state-of-the-art FP analysis when the security constraint has reasonable timing penalties.

key words: *cyber-physical system, security, real-time scheduling, schedulability analysis*

1. Introduction

Embedded systems have been designed to perform specific functionalities and they have been evolving in very complex structures as computing and control technology have developed. Such physical systems need to meet various requirements simultaneously, including high performance, real-time functioning, and a high level of reliability and security. A cyber-physical system (CPS) is a computer system that controls multiple physical devices. A unmanned aerial vehicle surveillance system (UAV) is a compelling example of a CPS because it manipulates various physical devices to navigate a route avoiding obstacles, to control surfaces to maintain flight, to adjust propulsion speeds, and to process video streams. A CPS executes such tasks in a timely manner with a real-time scheduling algorithm to meet its deadlines.

As many CPSs manipulate security-sensitive data, enhancing the quality of security in CPSs is a very critical and challenging issue in CPS design [1]. Although there has been a large body of research related to general purpose PCs, applying such techniques that provide confidentiality, integrity protection and availability to CPS is not an effective approach. As timely execution of tasks in CPS typically relies on real-time scheduling, direct adoption of existing security techniques in CPSs may compromise the real-time property of CPSs. Thus, in securing CPSs, satisfying the real-time requirement is a fundamental issue that should be considered in CPS design.

To cope with such issues, some previous works have proposed approaches to add a security constraint to the real-time properties [2]–[4]. In particular, some researches have focused on maximizing the level of security while meeting real-time deadlines by improving earliest deadline first (EDF) scheduling [2], [4]. These approaches have advantages over other techniques. Because they are software based approaches, they are easier to deploy than hardware based approaches, they do not require many additional resources and the existing analytical tools for real-time systems allow analysis of how security requirements influence the real-time property because security is incorporated as a part of the constraint of real-time scheduling.

Taking advantage of such approaches, several researches resolved the information leakage problem that can arise between real-time tasks with different security levels [5], [6]. It is well known that information leakage can occur between tasks sharing resources without explicit communication [7], [8]. A previous work [6] suggested a pre-flush mechanism in which the state of shared resources such as cache, DRAMs and even I/O bus are flushed if there is a possibility of information leakage after a real-time task finishes its execution. It also proposed an improved fixed priority (FP) [9] scheduler incorporating a security constraint and schedulability analysis techniques that are aware of the pre-flush task to reduce the potential for information leakage via shared resources. However, this mechanism is limited to non-preemptive scheduling [10] and there is still room to narrow the wide gap between the analytic result and the result of scheduling.

In this paper, we present a new preemptive fixed priority scheduling algorithm incorporating a security constraint, called lowest security-level first (LSF) and its schedulability analysis to reduce the potential of information leakage. LSF assigns higher priority to a task that has a lower security level. We also discuss why conventional techniques can not be directly applied to the scheduling model considering both security and real-time constraint to resolve information leakage problem, and its analytic results have not

Manuscript received December 9, 2015.
Manuscript revised February 25, 2016.
Manuscript publicized April 22, 2016.

[†]The authors are with School of Computing, KAIST, 219 Daehak-ro, Yeseong-gu, Daejeon, 305–701 Korea.
[††]The author is with the Dept. of Computer and Information Science, University of Pennsylvania, USA.
a) E-mail: jaewoo@cis.upenn.edu (Corresponding author)

DOI: 10.1587/transinf.2015EDP7493

been tightly bounded. Our experimental results show that, although LSF is not a better performing real-time scheduling algorithm than rate monotonic (RM) [11], its exact analysis outperforms the schedulability analysis of RM when the security constraint has reasonable timing overheads.

The contributions of our work are the following:

- We present the LSF scheduling algorithm inducing a strong (exact) schedulability analysis.
- We investigate how the timing overhead incurred due to security constraint influences the schedulability of a real-time system in terms of real-time scheduling. Based on the investigation, we propose a new pre-flush mechanism, called *flush task reservation* to invoke pre-flushing tasks not in a greedy manner.
- Using the combination of LSF scheduling algorithm and *flush task reservation*, we derive strong schedulability condition and suggest an algorithm for schedulability analysis.
- We conducted experiments to compare our techniques with conventional approaches and we discuss how variation of the timing overhead value for the security constraint affects the performance of each technique.

2. Adversary and System Models

We follow the adversary model and the system model proposed in [6] derived from actual examples of real-time systems such as an avionics system designed by the DO-178B model [12] and the "RePLACE" system of Northrop Grumman [13]–[15]. Operation of some sub-systems in these systems depends on another sub-system designed by a less trustworthy vendor, which can cause an information leakage problem. For example, the navigation system in an avionics system can be made by less trustworthy vendor since it is less critical than the flight control system. In this case, the compromised navigation system can glean sensitive data from shared resources while the navigation system and the flight control system are scheduled according to the given scheduler and actively communicate with each other.

2.1 Adversary Model

We assume that an adversary can insert new tasks and compromise some tasks. We also assume that the inserted task can consider real-time guarantee so that the task cannot be detected immediately. The major goal of an attacker is to obtain the information on the shared resource that multiple tasks use. Adversaries that can tamper with the system operation are out of scope for this work.

2.2 System Model

We consider a uni-processor system following Liu and Layland's task model [11] containing a sporadic task set τ where a task $\tau_i \in \tau$ is specified as (T_i, C_i, D_i) such that T_i is the minimum inter arrival time, C_i is the worst-case execution time requirement, and D_i is the relative deadline ($C_i \leq D_i \leq T_i$). Since considering the exact cost of preepmtion is out of this paper, we assume that the worst-case execution time C_i includes the preemption cost. Here, S_i is defined as the security level of a task τ_i. A greater number of S_i means a higher security level of τ_i. Task τ_i releases its instance called job periodically and j_i^k denotes the k^{th} job of the task. Each job j_i^k is released at r_i^k. Here, R_i^k and R_i are defined as the response time of j_i^k and the worst-case response time of τ_i respectively.

3. Security and Scheduling

To motivate our works, let us take a simple example considering two real-time tasks; a task τ_H with high priority and a high security level, and a task τ_L with low priority and a low security level. Let us consider a flight control module for τ_H and a sensing module for τ_L in an UAV as examples. Since τ_H has a higher security level than τ_L, information from τ_H should not leak to τ_L. These two tasks share resources, such as cache, and are scheduled with a fixed priority scheduling algorithm. According to the priority ordering, τ_L has a high probability of being executed immediately after the execution of τ_H. Assume that an instance of τ_L is released but cannot be executed due to a job of τ_H being executed, or a job of τ_L is preempted from a newly released job of τ_H. If τ_L executes immediately after τ_H, information leakage can occur because τ_L can inspect the cache and obtain sensitive information from it.

Flushing the state of shared resources previously used by other tasks can be a solution for preventing many attacks exploiting this vulnerability. In the case of cache, we need some time for the flushing task to remove information on the shared cache after execution of τ_H. For a disk, flushing can move head to some initial positions to avoid exposure of the previous state of the disk. The I/O bus is another example. When a new task is invoked, the I/O bus can carry the information of a previous task [16]. In this case, timing delay can be used to complete the previous task. In general, adopting the flushing task mechanism on shared resource requires timing overhead and a real-time scheduler should be aware of such overhead for effective scheduling and schedulability analysis.

3.1 Flush Task Mechanisms

The real-time scheduler and schedulability analysis can apply a security constraint to address the information leakage problem; *before τ_L is scheduled, a timing penalty should be spent to flush out the state of shared resources after execution of τ_H*. To satisfy this constraint, we invoke a flush task which should spend some execution time at all transitions, $\tau_H \rightarrow \tau_L$. The flush task has the highest priority, and it flushes information on shared resource so that a following task cannot access previous information on shared resources recently used. We also can assume invocation of flush tasks at all transitions, $\tau_L \rightarrow \tau_H$. However, information leakage

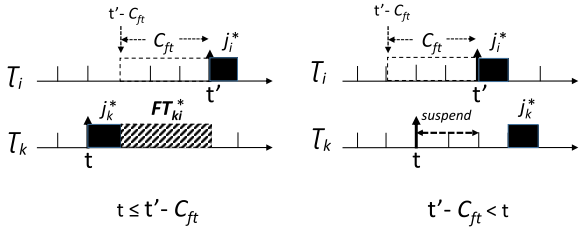


Fig. 1 Two cases of flush task reservation.

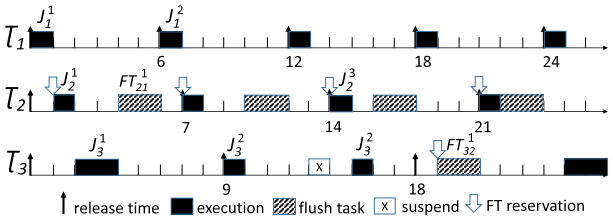


Fig. 2 LSF exmaple.

Table 1 Task parameters of LSF example.

	T_i	C_i	S_i
τ_1	6	1	1
τ_2	7	1	2
τ_3	9	2	3

from τ_L to τ_H is not as critical as that from τ_H, and we can increase utilization by not considering such an assumption. Thus, we do not consider this assumption.

As we mentioned in the previous section, LSF assigns a higher priority to a task that has a lower security level. In addition, when a job is released, the LSF scheduler identifies the time when the earliest higher priority job will be released and reserves flush task invocation avoiding interference to the release of the higher priority job. We call this mechanism *flush task reservation*. Figure 1 illustrates how *flush task reservation* works for two cases. For a job j_k^* just released or resumed its execution after preemption from any higher priority job at time t, let us assume that the earliest higher priority job j_i^* will be released at time $t' > t$. If $t \leq t' - C_{ft}$ (the case of Fig. 1 (a)), where C_{ft} is the execution time of flush task, LSF scheduler reserves a flush task FT_{ki}^* to prevent information leakage from j_k^* to j_i^* so that FT_{ki}^* will be invoked at the time $t' - C_{ft}$. If $t > t' - C_{ft}$ (the case of Fig. 1 (b)), the execution of j_k^* is suspended in the interval between t and t' so that FT_{ki}^* cannot interfere τ_i. In both cases, security constraint is satisfied with FT_{ki}^* and j_i^* can execute without any interference from FT_{ki}^* reserved and invoked by LSF scheduler.

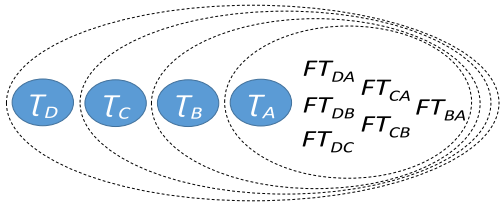
Unlike LSF using *flush task reservation*, conventional scheduling algorithms [6] adopt a greedy approach for flush task invocation. When a job is released, a flush task is invoked if the latest finished job has a higher security level. This naive mechanism can cause interference from the flush task to the higher priority task because a flush task is invoked without consideration of release time of higher priority tasks. If we adopt existing naive approach for the the case of Fig. 1 (b), the execution of j_i^* might be hindered by FT_{ki}^* invoked after execution of j_k^* even though j_i^* has higher priority than j_k^*. Thus, we derive a property of conventional scheduling policy that the execution of higher priority job is influenced by the behaviour of lower priority job.

Example. Figure 2 describes that tasks with parameters defined in Table 1 are scheduled according to LSF scheduler with $C_{ft} = 2$. According to the security level ordering described in Table 1, τ_1 has the highest priority and τ_3 has the lowest priority among tasks. As shown in Fig. 2, the first flush task FT_{21}^1 is reserved at time 1 when the first job j_2^1 of τ_2 starts to execute, and is invoked at time 4 avoiding interference to j_1^2 since the second job j_1^2 of τ_1 will be released at time 6. As *flush task reservation* policy is applied, all flush tasks FT_{21}^* to prevent information leakage from τ_2 to τ_1 are invoked without interfering execution of all jobs of τ_1. Since τ_3 has the lowest priority, all flush tasks FT_{31}^* and FT_{32}^* should consider release time of jobs of both τ_1 and τ_2. To prevent information leakage from job j_3^1 to the job j_1^2, a flush task should be reserved since j_1^2 will be released at time 6, which has lower security level. However, LSF does not need to reserve a flush task FT_{31}^1, since FT_{21}^1 is already reserved and will be invoked at time 4 and it also prevents information leakage from j_3^1 to j_1^2. This is a special case of Fig. 1 (a). At time 13, the job j_3^2 is ready to be executed but its execution is suspended. If j_3^2 executes at time 13, it should be preempted by j_2^3 at time 14. If so, FT_{32}^1 should be invoked before execution of j_2^3, which can interfere j_2^3. To avoid such interference, j_3^2 suspends its execution and resumes it at time 15. This is an example of Fig. 1 (b) case. As we can observe in Fig. 2, all jobs are not interfered from both execution of its lower priority jobs and flush tasks related to lower priority jobs thanks to *flush task reservation* mechanism.

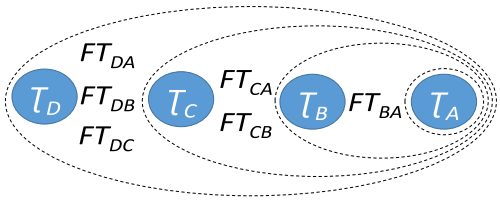
3.2 Interference Relations of Flush Task Mechanisms

Figure 3 compares interference relation of tasks scheduled by LSF using *flush task reservation* with greedy flush task invocation that conventional scheduling algorithms have adopted. We consider four tasks τ_A, τ_B, τ_C and τ_D with priority ordering $\tau_D < \tau_C < \tau_B < \tau_A$, where $\tau_B < \tau_A$ denotes that τ_A has higher priority than τ_B. We assume that tasks are scheduled with the LSF scheduling algorithm. Here, FT_{ij} denotes flush task invocation between a job of τ_i and that of τ_j to prevent information leakage from τ_i to τ_j. An ellipse drawn with a dashed line represents a interference relation meaning that only entities in the same ellipse can interfere with each other according to their scheduling priorities. In other words, entities outside the ellipse can not interfere with entities within the ellipse. In addition, an entity can be interfered with entities inside the ellipse that contains the entity according to the scheduling priority of each entity.

As seen in Fig. 3 (a), all tasks are subject to interference by both higher priority tasks and any kind of flush task FT_{ij}



(a) conventional flush task invocation case



(b) flush task reservation case

Fig. 3 Examples of interference relations with *flush task reservation* and conventional flush task invocation for priority ordering, $\tau_D < \tau_C < \tau_B < \tau_A$.

in the conventional flush task invocation case. In the case of τ_A, it is not subject to interference from any other task because the tasks are scheduled with a preemptive scheduling algorithm and τ_A has the highest priority among the tasks. However, execution of τ_A can be hindered by any kind of flush task that is executed with the highest priority. This is because a flush task is invoked in a greedy manner without consideration of the release times of higher priority jobs as we mentioned previously.

In a *flush task reservation* case (see Fig. 3 (b)), however, all tasks are only subject to interference from higher priority tasks and flush tasks related to higher priority tasks. For example, τ_B of the *flush task reservation* case is only subject to interference from τ_A and FT_{BA} but not from others. Thanks to the policy of *flush task reservation*, all flush tasks except FT_{BA} should be reserved and invoked without interfering with the release of jobs of τ_A or τ_B. As seen in Fig. 3 (b) this property is applied for all tasks. Thus, it is guaranteed that all jobs are not subject to interference from the behavior of lower priority jobs, including the execution of flush tasks related to lower priority jobs.

Schedulability analysis for the scheduling algorithm using a conventional flush task invocation mechanism is exclusively sophisticated. This is because the execution of higher priority jobs is influenced by behavior of lower priority jobs that can invoke flush tasks that can interfere with higher priority jobs. Since most strong schedulability analyses for preemptive systems are based on the property that higher priority jobs must not be influenced by the behavior of lower priority jobs [10], [11], [17], those techniques cannot be applied directly for scheduling algorithms incorporating security constraints. Thus, existing schedulability analysis tests bound the number of FT invocations in a very naive manner. LSF using *flush task reservation* not only resolves this problem by reserving the property of preemptive systems but also allows strong schedulability analysis to be adapted to the LSF scheduler incorporating security constraints.

4. Definitions for LSF Schedulability Analysis

Based on the approach we proposed, each task τ_i only needs to consider interference from higher priority tasks and flush task invocation among those tasks for its test. Thus, we define *hyperperiod at level i*, defined as $H_i = LCM\{T_j\}_{\tau_j \in hep(\tau_i)}$, where $hep(\tau_i)$ is the set of tasks with a priority higher than or equal to task τ_i and LCM$\{T_j\}$ is the least common multiple of T_j. τ_i releases its job σ_i times in hyperperiod at level i, where σ_i is described as

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

We define T_i-*mesoid* as a ordered set which describes the state of each job j_i^k. We denote a time unit already executed or suspended by an "e", which is called *consumption*, a time unit still available by an "a" and a time unit for a flush task invocation by "f". For example, $\{e, e, e, a, a, f, f, e, e, e, e, e, a, a\}$ is a mesoid where the first 3 time units have already been executed, the next 2 time units are available, followed by 2 for a flush task invocation, then 5 executed time units followed by 2 available time units. We present consumptions by its cardinal inside brackets (c), with $c \in \mathbb{N}^+$. We use another brackets $[r]$, with $r \in \mathbb{N}^+$ for execution time of flush tasks to distinguish with brackets (c). In addition, we enumerate the sequence of available time units according to the natural numbers. Each of these natural numbers is called an *availability*. Based on the description, the previous 14-mesoid can be re-written as $\{(3)1, 2, [2], (5), 3, 4\}$. It has two consumption $3, 5$, a reservation 2, and four availabilities $1, 2, 3, 4$. Since each T_i-*mesoid* describes the state of each job, there are σ_i T_i-mesoids in H_i which are sequences of T_i-mesoids. We define $\mathcal{L}_i^b = \{\mathcal{M}_i^{b,1}, \mathcal{M}_i^{b,2}, \cdots, \mathcal{M}_i^{b,\sigma_i}\}$ as the sequence of σ_i T_i-mesoids before τ_i is scheduled. For example, $\mathcal{L}_i^b = \{\{(5), 1, 2, [1], (2), 3\}\{1, [1], (3), 2, [1], (3), 3\}\}$ is a sequence of $\sigma_i = 2$ 11-mesoids.

We define the cell of the $\mathcal{L}_i^b$ as a set of time units between adjacent two consumptions. For the above example, we have

$$\mathcal{L}_i^b = \{\{(5)\overbrace{1, 2, [1]}^{[1]^i}, (2), \overbrace{3}^{[2]^i}\}, \overbrace{1, [1]}^{[2]^i}, (3), \overbrace{2, [1]}^{[3]^i}, (3), \overbrace{3}^{[4]^i}\}\}$$

where for $m \in \mathbb{N}$ $[m]^i$ denotes the m^{th} cell in $\mathcal{L}_i^b$. We call $\mathcal{X}_i^k$ universe of τ_i to be the set which consists of all the availabilities of $\mathcal{M}_i^{b,k}$. For the previous example of a sequence of 11-mesoids, $\mathcal{M}_i^{b,2} = \{1, [1], (3), 2, [1], (3), 3\}$, and thus we have $\mathcal{X}_i^2 = \{1, 2, 3\}$.

$\mathcal{L}_i^b$ shows that the execution time of the first job j_i^1 and the second job j_i^2 should not exceed 3 otherwise task τ_i cannot be schedulable. We call $\mathcal{L}_i^a = \{\mathcal{M}_i^{a,1}, \mathcal{M}_i^{a,2}, \cdots, \mathcal{M}_i^{a,\sigma_i}\}$

the sequence of σ_i T_i-mesoids of task τ_i after τ_i is scheduled. $\mathcal{L}_i^a$ is resulted from $\mathcal{L}_i^b$ as the available time units will have been consumed up to the response time within each mesoid $\mathcal{M}_i^{b,k}, k = 1, \cdots, \sigma_i$ of task τ_i after τ_i is scheduled. The process of building $\mathcal{L}_i^a$ from $\mathcal{L}_i^b$, and $\mathcal{L}_{i+1}^b$ from $\mathcal{L}_i^a$ respectively will be detailed in the next section.

To sum up, for every task τ_i, we have

$$\tau_i : \begin{cases} \mathcal{L}_i^b = \{\mathcal{M}_i^{b,1}, \mathcal{M}_i^{b,2}, \cdots, \mathcal{M}_i^{b,\sigma_i}\} \\ \mathcal{L}_i^a = \{\mathcal{M}_i^{a,1}, \mathcal{M}_i^{a,2}, \cdots, \mathcal{M}_i^{a,\sigma_i}\} \end{cases}$$

Note that both $\mathcal{L}_i^b$ and $\mathcal{L}_i^a$ contain σ_i T_i-mesoids.

5. LSF Schedulability Analysis

This section derives a schedulability condition for the proposed LSF scheduling algorithm using *flush task reservation*. We first describe how schedulability is tested for two tasks case to motivate our result. And then we generalize it to result stronger schedulability condition than proposed in [6].

5.1 Scheduling of Two Tasks

To generalize our schedulability analysis process for n tasks, we first present the analysis sequence with two tasks, $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$, with $S_1 < S_2$ which means τ_2 has a higher security level than τ_1. Under LSF scheduling, τ_1 is assigned higher priority because τ_1 has a lower security level. Because τ_1 has the highest priority, each job j_1^* of τ_1 can consume any availability in T_i and is never preempted. Since it does not suffer any interference, its response time is equal to C_1, which results in C_1 consumptions and T_1-C_1 availabilities in each instance. Thus, T_1-mesoids of τ_1 are described as

$$\tau_1 : \begin{cases} \mathcal{L}_1^b = \{\mathcal{M}_1^{b,1}\} = \{\{1, 2, \cdots, T_1\}\} \\ \mathcal{L}_1^a = \{\mathcal{M}_1^{a,1}\} = \{\{(C_1), 1, 2, \cdots, T_1 - C_1\}\} \end{cases}$$

The next step is to schedule task τ_2 by taking into account the cost of flush tasks FT_{21}^* invoked to prevent information leakage from τ_2 to τ_1. The building process of $\mathcal{L}_2^b$ consists of $\sigma_2 = \frac{H_2}{T_2}$ T_2-mesoids. The construction is done with two phases; inheritance phase and *flush task reservation* phase.

In the first phase, $\mathcal{L}_2^b$ inherits the time units of $\mathcal{L}_1^a$ $\frac{H_2}{T_1}$ times so that all availabilities in $\mathcal{L}_2^b$ are filled with the time units of $\mathcal{L}_1^a$ in a cyclic manner. In other words, $(x * T_1 + y)^{th}$ time unit in $\mathcal{L}_2^b$ is filled with y^{th} time unit in $\mathcal{L}_1^a$, where $0 \le x < \frac{H_2}{T_1}$, and $1 \le y \le$ the number of time units in $\mathcal{L}_1^a$. After the construction of $\mathcal{L}_2^b$, we can easily determine the corresponding universe $\mathcal{X}_2^k$ to each T_2-mesoid $\mathcal{M}_2^{b,k}$, $k = 1, \cdots, \sigma_2$.

Let us take the following example in order to illustrate the process of building $\mathcal{L}_2^b$ from $\mathcal{L}_1^a$. We consider a set of two tasks τ_1 and τ_2 with $T_1 = D_1 = 6$, $T_2 = D_2 = 8$, and $C_1 = 2, C_2 = 1$. We obtain

$$\tau_1 : \begin{cases} \mathcal{L}_1^b = \{\mathcal{M}_1^{b,1}\} = \{\{1, 2, 3, 4, 5, 6\}\} \\ \mathcal{L}_1^a = \{\mathcal{M}_1^{a,1}\} = \{\{(2), 1, 2, 3, 4\}\} \end{cases}$$

Since $\sigma_2 = \frac{H_2}{T_2} = 3$, we construct $\mathcal{L}_2^b$ which consists of a sequence of three 8-mesoids by using the mechanism inheriting time units of $\mathcal{L}_1^a$ described previously. Based on the mechanism, we have

$$\mathcal{L}_2^b = \{\mathcal{M}_2^{b,1}, \mathcal{M}_2^{b,2}, \mathcal{M}_2^{b,3}\} = \{\{(2), 1, 2, 3, 4, (2)\}, \{1, 2, 3, 4, (2), 5, 6\}, \{1, 2, (2), 3, 4, 5, 6\}\}$$

With the definition of *cell*, $\mathcal{L}_2^b$ can be also be written as

$$\mathcal{L}_2^b = \{\{(2) \overbrace{1, 2, 3, 4}^{[1]^2}, (2)\}, \overbrace{\{1, 2, 3, 4\}}^{[2]^2}, (2), \overbrace{5, 6\}, \{1, 2}^{[3]^2}, (2), \overbrace{3, 4, 5, 6\}}^{[4]^2}\}$$

We now perform *flush task reservation* phase to prevent information leakage at all transitions $\tau_2 \to \tau_1$. The combination of LSF and *flush task reservation* mechanism guarantees that execution of flush task is finished at the right corner of each *cell* and execution of each job of τ_2 starts at the left corner of *cell* without flush task invocation at transitions $\tau_1 \to \tau_2$. This is because LSF using *flush task reservation* guarantees that higher priority task must have lower security level and flush task is invoked at $t' - C_{ft}$ (see Fig. 1) to avoid interference to higher priority task. After *flush task reservation* phase of our analysis, $\mathcal{L}_2^b$ with $C_{ft} = 1$ can be written as

$$\mathcal{L}_2^b = \{\{(2) \overbrace{1, 2, 3, [1]}^{[1]^2}, (2)\}, \overbrace{\{1, 2, 3, [1]\}}^{[2]^2}, (2), \overbrace{4, 5\}, \{1, [1]}^{[3]^2}, (2), \overbrace{2, 3, 4, [1]\}}^{[4]^2}\}$$

For each 8-mesoid $\mathcal{M}_2^{b,k}$, $1 \le k \le 3$, composing $\mathcal{L}_2^b$, we build the corresponding universe $\mathcal{X}_2^k$, $1 \le k \le 3$. These universe are given by

$$\tau_2 : \begin{cases} \mathcal{X}_2^1 = \{1, 2, 3\} \\ \mathcal{X}_2^2 = \{1, 2, 3, 4, 5\} \\ \mathcal{X}_2^3 = \{1, 2, 3, 4\} \end{cases}$$

Now each job j_2^k of τ_2 is ready to execute without any information leakage and invocation of flush task hindering the execution of higher priority tasks. The amount of C_2 availability in each universe $\mathcal{X}_2^k$, $1 \le k \le \sigma_2$ are consumed by execution. The response time R_2^k, $1 \le k \le \sigma_2$ of task τ_2 in its k^{th} job is computed by summing consumptions of higher priority tasks and execution of flush task faced before j_2^k finishes its execution. As $C_2 = 1$ and first job j_2^1 of τ_2 is the only one interfered from the execution of higher priority job, response times of j_2^1, j_2^2 and j_2^3 are $R_2^1 = 3$, $R_2^2 = 1$, and $R_2^3 = 1$. Then, the worst-case response time R_2 of task τ_2 is obtained by

$$R_i = max_{\{1 \le k \le \sigma_i\}}(R_i^k) \tag{2}$$

With computed worst-case response time R_2, τ_2 is schedulable if and only if

$$R_i \leq D_i \tag{3}$$

Based on the Eq. (2), the worst-case response time R_2 is 3 and it satisfies schedulability condition described by Eq. (3). Thus, we verify that task τ_2 is schedulable. For more than two tasks, we derive $\mathcal{L}_2^a$ deduced from $\mathcal{L}_2^b$ as follows

$$\mathcal{L}_2^b = \{\{(3), 1, 2, [1], (2)\}, \{(1), 1, 2, [1], (2), 3, 4\}, \{(1), [1], (2), 1, 2, 3, [1]\}\}$$

5.2 Scheduling of $n > 2$ Tasks

The analysis test algorithm for more than two tasks is conducted by a mechanism similar to that described in the previous section. Thanks to the property of LSF using *flush task reservation*, testing of a task τ_i does not take into account the behavior of lower priority tasks including their execution and flush tasks associated with them.

Algorithm 1 presents our schedulability test in detail. 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. LSF analysis first compute hyperperiod at level i and the number of instance within hyperperiod (Line 2 in Algorithm 1). Then it builds T_i-mesoids where all time units are available composing $\mathcal{L}_i^b$ (Lines 3–5 in Algorithm 1). The sequence $\mathcal{L}_i^b$ is derived from $\mathcal{L}_{i-1}^a$ with inheritance phase (Lines 6–10 in Algorithm 1) and *flush task reservation* phase (Lines 11–20 in Algorithm 1). Then we can easily determine the universes $\mathcal{X}_i^k \; \forall k \in 1, \cdots, \sigma_i$ from availabilities in $\mathcal{L}_i^b$ (Lines 21–23 in Algorithm 1). After construction of $\mathcal{L}_i^b$, schedulability of τ_i is tested with Eq. (3). If τ_i is schedulable, $\mathcal{L}_i^a$ will be built based on the sequence $\mathcal{L}_i^b$ and the amount of execution C_i consuming availabilities in universes $\mathcal{X}_i^k$, for the schedulability test of τ_{i+1} (Lines 25–32 in Algorithm 1). If all tasks satisfy the schedulability condition, then we conclude that the task set is schedulable.

Complexity. As LSF schedulability analysis is a per job test, the time complexity of the analysis includes the number of jobs in a hyperperiod at level i. To compute the exact amount of interference from higher priority jobs and flush task invocations to each job, we need to consider the number and positions of available time units in advance. Then we take into account the number of higher priority jobs in each mesoid since the number of potential flush task invocation is equal to the number of higher priority tasks. Thus the complexity of LSF schedulability analysis is $O(\sigma_i \cdot \sigma_{hp} \cdot m_0)$ where, σ_i is the number of jobs and σ_{hp} is the number of higher priority jobs in a hyperperiod at level i, and m_0 is the number of time units in the sequence of mesoids of each task.

Since the time complexity of LSF schedulability analysis adds a factor m_0, the time required for the analysis can be longer if the least common multiple of periods of tasks is a very large value. In this case, we can use the least common multiple reduction mechanism under an assumption that reduced periods do not degrade the functionality of the target systems. The experimental results of the common multiple reduction mechanism proposed in [18] showed that least common multiples derived by 20 sets of five randomly generated periods taking values in the interval [0, 1000] reduced by a factor of 112,722,685 on average. Also, the mechanism bounds the least common multiples of the five periods by 151,200 which is an acceptable value for schedulability analysis. It is worth noting that for the harmonic sequence of periods of tasks, the time complexity of LSF schedulability analysis is $O(n)$ where n is the number of tasks in the task set.

Algorithm 1 LSF Schedulability Analysis

1: **for** $i = 2$ to n **do**
2: $\quad\sigma_i \leftarrow \frac{H_i = LCM\{T_j\}_{\tau_j \in hep(\tau_i)}}{T_i}$
3: $\quad$**for** $x = 1$ to σ_i **do**
4: $\quad\quad$Build empty $\mathcal{M}_i^{b,x}$ composing $\mathcal{L}_i^b$
5: $\quad$**end for**
6: $\quad$**for** $x = 0$ to $\frac{H_i}{T_{i-1}}$-1 **do**
7: $\quad\quad$**for** $y = 1$ to the number of time units in $\mathcal{L}_{i-1}^a$ **do**
8: $\quad\quad\quad$$((x * T_{i-1} + y)^{th}$ time unit in $\mathcal{L}_i^b) \leftarrow (y^{th}$ time unit in $\mathcal{L}_{i-1}^a)$
9: $\quad\quad$**end for**
10: $\quad$**end for**
11: $\quad$Build all cells in $\mathcal{L}_i^b$
12: $\quad$**for** $m = 1$ to the number of cells in $\mathcal{L}_i^b$ **do**
13: $\quad\quad$**if** $[m]^i$ does not contain a flush task **then**
14: $\quad\quad\quad$**if** $[m]^i$ is large enough to contain a flush task **then**
15: $\quad\quad\quad\quad$a flush task is reserved in $[m]^i$
16: $\quad\quad\quad$**else**
17: $\quad\quad\quad\quad$all availabilities in $[m]^i$ are consumed
18: $\quad\quad\quad$**end if**
19: $\quad\quad$**end if**
20: $\quad$**end for**
21: $\quad$**for** $x = 1$ to σ_i **do**
22: $\quad\quad$Build $\mathcal{X}_i^x$ from availabilities in $\mathcal{M}_i^{b,x}$
23: $\quad$**end for**
24: $\quad$Compute worst-case response time of τ_i and test schedulability of τ_i with Eq. (3)
25: $\quad$**if** τ_i is deemed schedulable according to schedulability test **then**
26: $\quad\quad$**for** $x = 1$ to σ_i **do**
27: $\quad\quad\quad$first C_i units in $\mathcal{X}_i^x$ are consumed
28: $\quad\quad$**end for**
29: $\quad$**else**
30: $\quad\quad$return unschedulable
31: $\quad$**end if**
32: $\quad$Build $\mathcal{L}_i^a$ by duplicating $\mathcal{L}_i^b$
33: **end for**
34: return schedulable

6. Evaluation

In this section, we evaluate our approaches by simulation. We first describe the simulation environment, including the task generation method and task parameters. Then, we discuss the performance of the LSF scheduling algorithm and

Table 2 Parameters for experiments.

Parameter	Value
Number of tasks, N	$N \in \{3, 4, \cdots, 10\}$
Task period, p_i	$p_i \in \{50, 100, \cdots, 1000\}$
Task execution time, e_i,	$e_i \in \{5, 6, \cdots, 50\}$
Security level of task, S_i,	$S_i \in \{3, 4, \cdots, 10\}$
FT overhead, C_{ft}	$C_{ft} \in \{1, 5, 10\}$

its schedulability test compared to conventional techniques.

6.1 Simulation Environment

We randomly generated 2000 tasks based on the task set generation method used in [6]. We set 10 base utilization groups $[0.02 + 0.1 \cdot i,\ 0.08 + 0.1 \cdot i]$ for $i = 0, \cdots, 9$, *i.e.*, 200 instances per group. 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. Base utilization of an instance is defined as the total sum of task utilizations. Table 2 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 period $p_i \in \{50, 100, \cdots, 950, 1000\}$ and an execution time $e_i \in \{5, 6, \cdots, 49, 50\}$. The deadline of each task is equal to its period. Each task has security level $S_i \in \{3, 4, \cdots, 9, 10\}$ avoiding cyclic ordering of security levels of tasks in a task set. The overhead of flush task (FT) has a value in the set of $\{1, 5, 10\}$. The values are derived on the actual resources in actual architecture (*e.g.* in the case of a cache, $\frac{size\ of\ cache}{cache\_refill\_bandwith}$ from the core i7 or Tegra 3 device) as described in [6].

6.2 Simulation Results

We chose an RM scheduling algorithm as a base line, since it is a well known optimal preemptive fixed priority scheduling algorithm for Liu and Layland's task model showing high performance on a uni-processor.

The following scheduling algorithm and schedulability tests were considered:

- the preemptive RM scheduling algorithm using conventional FT invocation (RM – P)
- the preemptive LSF scheduling algorithm using *flush task reservation* (LSF – P)
- the non-preempitve RM analysis with the obviously bounded number of FT invocations proposed in [6] (RM – ob – NP)
- the RM analysis with the number of FT invocations bounded by Max-Flow approach for non-preemptive RM scheduling proposed in [6] (RM – maxflow – NP)
- the RM analysis with the obviously bounded number of FT invocations for preemptive RM scheduling, proposed in [6] (RM – ob – P)
- the LSF analysis with the obviously bounded number of FT invocations for preemptive LSF scheduling (LSF – ob – P)
- our LSF test with the exactly bounded number of FT invocations for preemptive LSF scheduling (LSF – ours – P)

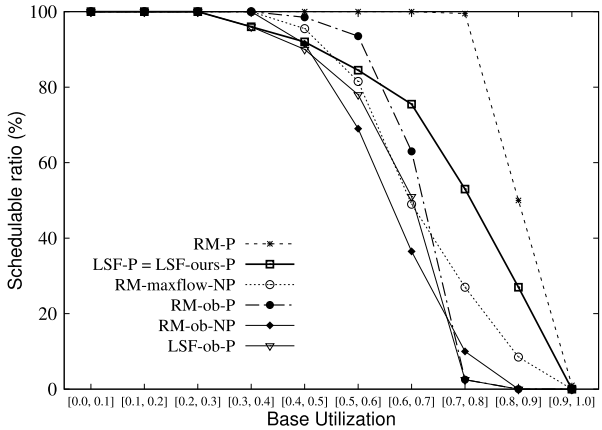


Fig. 4 Experimental result for $C_{ft} = 5$.

All analysis except LSF-ours-P use response time analysis with the bounded number of FT invocations. The number of FT invocations N_{ft} of non-preemptive and preemptive systems can be bounded by taking account the number of preemptions in each system [6]. For the former, $N_{ft} = N_{hepi} + 1$ and for the latter, $N_{ft} = 2 * N_{hepi} + 1$ (N_{hepi} is the number of higher or equal priority jobs for each task τ_i). RM – ob – NP uses the former one and, RM – ob – P and LSF – ob – P use the latter one. RM – maxflow – NP bounds the number of N_{ft} tighter than the obvious approaches. Note that LSF-P and LSF-ours-P show the same performance as LSF-ours-P is exact analysis.

All simulated tasks in a task set execute from time 0 to the hyperperiod of the task set. The simulator determines the schedulability of a task set with the obtained response time of each task when the execution of all tasks is completed. The analysis engines calculate the worst-case response time of each task with its own approaches (*i.e.* in case of LSF-ours-P, we use Algorithm 1 described in the previous section). If all tasks in a task set are schedulable, the analysis engine deems that the given task set is schedulable.

We first evaluated our approach compared to state of the art techniques for C_{ft} of 5. The X-axis plots the utilization groups for the experiments while the Y-axis represents the total percentage of schedulable task sets for each bin. As seen in Fig. 4, RM-P performs well even though it is not carefully designed with consideration of the security constraint. However, RM-ob-P designed for RM-P, shows poor performance compared to RM-P due to its naive bound of the number of FT invocations. RM-maxflow-NP performs better than RM-ob-NP which uses naive bound of FT, but it is limited to non-preemptive scheduling, which frequently suffers from the priority inversion problem [19]. Due to the limitation, RM-maxflow-NP may not perform better than a naive approach, RM-ob-P for C_{ft} of 5. In terms of scheduling capability, LSF-P shows worse performance than RM-P for all base utilizations but in the exact analysis test, LSF-ours-P conditionally performs better than RM-ob-P. As seen

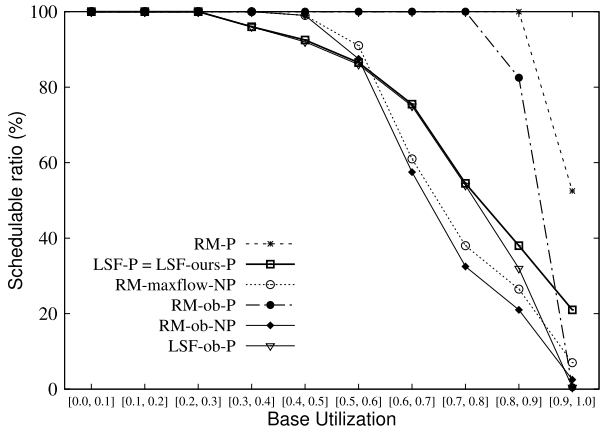


Fig. 5 Experimental result for $C_{ft} = 1$.

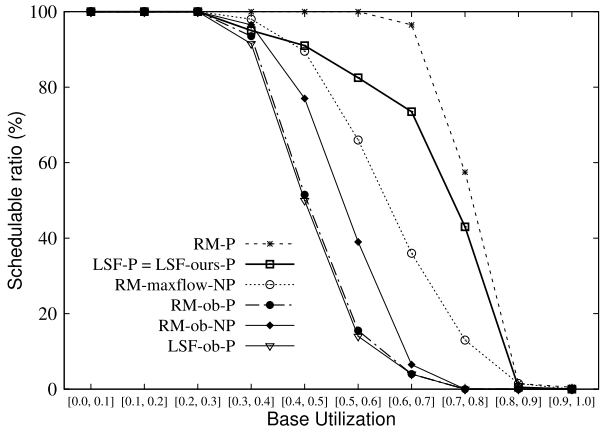


Fig. 6 Experimental result for $C_{ft} = 10$.

Table 3 Total percentage of schedulable task set of each technique and its performance comparisons.

	$C_{ft} = 1$	$C_{ft} = 5$	$C_{ft} = 10$
RM-P (%)	95	85	75
LSF-ours-P (%)	76	72	66
RM-ob-P (%)	88	66	46
RM-ob-P/RM-P	0.93	0.77	0.61
LSF-ours-P/LSF-P	1.0	1.0	1.0
LSF-P/RM-P	0.80	0.84	0.87
LSF-ours-P/RM-ob-P	0.86	1.09	1.42

in Fig. 4, the performance of RM-ob-P is better before base utilization is 0.65 then it drops rapidly after that. This is because the higher based utilization is, the more tasks and jobs a task set has. For RM-ob-P, overheads resulting from pessimistic bound of FT invocations become severe because the overheads in the analysis are proportional to the number of jobs in the tasks. In case of LSF-ours-P, however, it does not suffer from such problem thanks to its exact analysis capability. As shown in Fig. 4, LSF-ours-P performs best among the considered analysis techniques when the base utilization is higher than 0.65.

We also investigated the relation between the size of C_{ft} and the schedulability of each approach. As expected, the performance of scheduling algorithm and its analyses is degraded as C_{ft} increases. As shown in Fig. 5, RM-ob-P is close to RM-P when the overhead of FT is 1. This is because C_{ft} is very low compared to the execution times of tasks so it does not significantly affect to the schedulability of the system. Thus, it can be shown that RM-ob-P performs well even though it bounds the number of FT in a naive manner. However, its performance drops sharply as the FT overhead goes up. Although LSF-ours-P conditionally performs better than RM-ob-P according to the base utilization, the total percentage of the schedulable task set is higher than RM-ob-P when we take into account all base utilization for C_{ft} of 5. The total percentage of the schedulable task set of each technique is derived by the average of that of utilization bins. For C_{ft} of 10, the gap between two tests becomes larger (see Figs. 4 and 6).

Table 3 displays the total percentages of the schedulable task sets for all of the considered techniques and gives a detailed comparison of their performance according to variations of the FT overhead. As seen from the second row to fourth row in Table 3, the total percentages of the schedulable task sets of RM-P, LSF-ours-P, and RM-ob-P decrease with different dropping rates as C_{ft} increases from 1 to 10. In the case of RM-P, 95% of tasks are schedulable for $C_{ft} = 1$, while only 75% of tasks are schedulable for $C_{ft} = 10$. Due to the pessimistic FT bound of RM-ob-P, the performance of RM-ob-P decreases by 42% while RM-P drops 20% of its performance. For the C_{ft} of 10, RM-ob-P only gets 61% of predictability to its scheduling algorithm, RM-P while LSF-ours-P gets 100% of it (the fifth and sixth rows in Table 3). In the performance aspect of scheduling algorithms (the seventh rows in Table 3), LSF-P attains 80% of RM-P when C_{ft} is 1, and it reaches to 87% of RM-P when C_{ft} is 10. From the evaluation results, we can see that LSF-P well capturing properties of security constraint is less influenced by the increase in C_{ft}. In comparison of analysis tests (The last row of Table 3), LSF-ours-P performs worse than RM-ob-p for C_{ft}=1, but it produce 42% better results than RM-ob-P when C_{ft} is 10.

In general, our analysis technique outperforms conventional techniques when the FT overheads increases. This is because our techniques represented by LSF and *flush task reservation* well analyze how the security constraint affects the schedulability of real-time systems, especially regarding interference relations between tasks.

7. Related Work

There have been some studies dealing with the real-time requirement incorporating security mechanisms. Xie and Lin considered periodic task scheduling requiring security service which has varying overhead in relation to the level of service [2], [4]. They enhanced the conventional scheduler to satisfy the real-time requirement and proposed new scheduler while maximizing the level of security service. Mohan [6] resolved the information leakage problem on shared resource of real-time tasks, by incorporating the se-

curity constraint into the real-time requirement. They improved FP scheduling and schedulability analysis to prevent information leakage while satisfying both the real-time requirement and security constraint. However the proposed analysis techniques are limited to non-preemptive scheduling and there is a wide gap between the performance of scheduling algorithm and analysis result since the number of flush task invocation is not tightly bounded.

Some works have considered the information leakage problem in real-time database systems which have security constraints [20]–[22]. Son [21] proposed transaction scheduling and a concurrency control algorithm satisfying security and real-time requirements. This work included metrics to measure the fulfilment of the security requirement and the concurrency control algorithm allowing a trade-off between the security and real-time requirements [20]. Son [22] also proposed a notion of partial security to enable a trade-off between two.

Calculation of the number of flush task invocations is closely related to calculation of the number of preemptions of a given task. A previous work [23] proposed a mechanism to count exactly how many times the given task is preempted while the task is executed by fixed priority scheduling. When a task is preempted by a higher priority task that has a different security level, a flush task should be invoked before the preemption or after the preemption according to the security level of the higher priority task to prevent information leakage. Thus, the previous work gave many hints to bound the number of flush tasks in schedulability analysis. However, it did not consider the security constraint, and the mechanism is only applicable to fully preemptive scheduling, in which higher priority tasks should not be influenced by the behavior of lower priority tasks.

8. Conclusion

In this paper, we presented a new preemptive fixed priority scheduling algorithm called LSF and a schedulability analysis, which consider security constraint. We focused on understanding the costs involved in integrating a security constraint with the real-time property. We proposed a *flush task reservation* mechanism to lead strong schedulability analysis. Our experimental results demonstrated that LSF schedulability analysis outperforms state-of-the-art FP analysis when flush tasks have reasonable overhead.

Our long-term goal is to design real-time scheduling algorithms and schedulability analysis that are more suitable for CPSs. As mixed criticality is one of the main features of CPSs [24], we plan to extend our work to a mixed-criticality model in our future work. We also intend to consider other security issues such as availability and integrity.

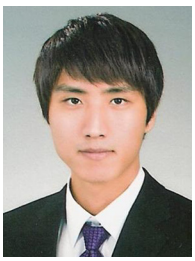
Acknowledgements

This research was supported by the National Research Foundation of Korea grant (NRF-2014R1A2A2A01006957) and 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, and No. 10041244, SmartTV 2.0 Software Platform).

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," 2010 IEEE Symposium on Security and Privacy (SP), pp.447–462, 2010
[2] M. Lin, L. Xu, L. 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, 2009.
[3] S.H. Son, R. Mukkamala, and R. David, "Integrating security and real-time requirements using covert channel capacity," IEEE Trans. Knowl. Data Eng., vol.12, no.6, pp.865–879, 2000.
[4] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," ACM Trans. Embed. Comput. Syst., vol.6, no.3, Article No.20, 2007.
[5] J. Son and J. Alves-Foss, "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in MLS systems," 2006 IEEE Information Assurance Workshop, pp.361–368, 2006.
[6] S. Mohan, M.K. Yoon, R. Pellizzoni, and R. Bobba, "Real-time systems security through scheduler constraints," 26th Euromicro Conference on Real-Time Systems (ECRTS), pp.129–140, 2014.
[7] P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," Advances in Cryptology –CRYPTO '96, Lecture Notes in Computer Science, vol.1109, pp.104–113, Springer, Berlin, Heidelberg, 1996.
[8] C. Percival, "Cache missing for fun and profit," Proc. BSDCan, 2005.
[9] J. Liu, Real-Time Systems, Prentice Hall, 2000.
[10] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," Software Engineering Journal, vol.8, no.5, pp.284–292, 2002.
[11] 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.
[12] European Organisation for Civil Aviation Electronics, "Do-178b: Software considerations in airborne systems and equipment certification," 1992.
[13] Northrop Grumman, "Replace," http://www.northropgrumman.com/Capabilities/RePLACE/Pages/default.aspx
[14] Northrop Grumman, "Reverse engineering for large applications." http://www.northropgrumman.com/Capabilities/RELA/Pages/default.aspx
[15] D. Reinhardt, "Certification criteria for emulation technology in the Australian defense force military avionics context," Proc. Eleventh Australian Workshop on Safety Critical Systems and Software, pp.79–92, 2006.
[16] M. Nam, R. Pellizzoni, L. Sha, and R. Bradford, "Integration support tool for real-time bus architecture designs," 14th IEEE International Conference on Engineering of Complex Computer Systems, pp.11–22, 2009.
[17] M.H. Klein, L. Sha, and J.B. Goodenough, "Rate monotonic analysis. technical report," report, 1991.
[18] J. Xu, "A method for adjusting the periods of periodic processes to reduce the least common multiple of the period lengths in real-time embedded systems," IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications (MESA), pp.288–294, 2010.
[19] J.A. Stankovic, M. Spuri, M.D. Natale, and G.C. Buttazzo, "Impli-

cations of classical scheduling results for real-time systems," Computer, vol.28, no.6, pp.16–25, 2002.
[20] Q. Ahmed and S. Vrbsky, "Maintaining security in firm real-time database systems," Proc. 14th Annual Computer Security Applications Conference, pp.83–90, 1998.
[21] S.H. Son, "Supporting timeliness and security in real-time database systems," 1997 Proc. Ninth Euromicro Workshop on Real-Time Systems, pp.266–273, 1997.
[22] S.H. Son, C. Chaney, and N.P. Thomlinson, "Partial security policies to support timeliness in secure real-time databases," IEEE Symposium on Security and Privacy, pp.136–147, 1998.
[23] P.M. Yomsi and Y. Sorel, "Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems," 19th Euromicro Conference on Real-Time Systems, ECRTS '07, pp.280–290, 2007.
[24] D. De Niz, L. Wrage, N. Storer, A. Rowe, and R. Rajkumar, "On resource overbooking in an unmanned aerial vehicle," Proc. 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems, ICCPS '12, pp.97–106, Washington, DC, USA, 2012.



Hyeongboo Baek is a Ph.D. student at the Department of Computer Science at KAIST, South Korea. He received B.S. degree in Computer Science and Engineering from Konkuk University, South Korea in 2010 and M.S. degree in Computer Science from KAIST, South Korea in 2012. His research interests include system security, cyber-physical systems, and real-time embedded systems. He won the best paper award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.

Jaewoo Lee received the B.S. and M.S. degrees in computer science and engineering from Seoul National University, in 2006 and 2008, respectively. He is currently a Ph.D. candidate in computer information science, University of Pennsylvania (expected graduation in 2016). His research interests include cyber-physical systems, real-time embedded systems, and model-driven engineering.

Yongjae Lee received his B.S. degree in computer science and engineering in 2008 from Chung-ang University and M.S. degree in 2010 from KAIST. He is currently working toward Ph.D. degree in computer science at KAIST. His research interests include system and network security.

Hyunsoo Yoon received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1979, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1981, and the Ph.D. degree in computer and information science from The Ohio State University, Columbus, in 1988. He is currently a Professor of with the Department of Computer Science, KAIST. During 1978–1980, he was with the Tongyang Broadcasting Company, Korea, then Samsung Electronics Company, Seoul, Korea, during 1980–1984. From 1988 to 1989, he was a Member of the Technical Staff with AT&T Bell Labs, Indial Hill, IL. Since 1989, he has been a Professor with the Department of Computer Science, Korea Advanced Institute of Science and Technology. His research interests include mobile ad hoc networks, wireless networks, and network security.