IEICE TRANS. INF. & SYST., VOL.E100–D, NO.9 SEPTEMBER 2017

PAPER

Incorporating Security Constraints into Mixed-Criticality Real-Time Scheduling

Hyeongboo BAEK[†], ***Member and*** **Jinkyu LEE**[†a)], ***Nonmember***

SUMMARY While conventional studies on real-time systems have mostly considered the real-time constraint of real-time systems only, recent research initiatives are trying to incorporate a security constraint into real-time scheduling due to the recognition that the violation of either of two constrains can cause catastrophic losses for humans, the system, and even environment. The focus of most studies, however, is the single-criticality systems, while the security of mixed-criticality systems has received scant attention, even though security is also a critical issue for the design of mixed-criticality systems. In this paper, we address the problem of the information leakage that arises from the shared resources that are used by tasks with different security-levels of mixed-criticality systems. We define a new concept of the security constraint employing a pre-flushing mechanism to cleanse the state of shared resources whenever there is a possibility of the information leakage regarding it. Then, we propose a new non-preemptive real-time scheduling algorithm and a schedulability analysis, which incorporate the security constraint for mixed-criticality systems. Our evaluation demonstrated that a large number of real-time tasks can be scheduled without a significant performance loss under a new security constraint.

key words: *mixed-criticality system, security, real-time scheduling, schedulability analysis*

1. Introduction

A Real-Time System (RTS) is a system being required to satisfy *real-time constraints* that every task on a RTS should completely performed its execution within its own deadline. As many RTSes manipulate security-sensitive data, and security violation in RTSes causes a greater catastrophic loss than those for general purpose systems, it is a critical issue in RTS design to enhance the quality of security in RTSes [1]. However, directly applying conventional security mechanisms for general purpose systems to RTSes can compromise real-time constraints since those mechanisms have not been designed with consideration of real-time constraints. In recognition of this property, recent studies tried to incorporate *security constraints* into real-time scheduling to capture the real-time and security constraints simultaneously [2]–[6].

A number of studies in [2]–[4] addressed the problem of information leakage that can arise on the shared resources used by real-time tasks with different security-levels in a RTS. The initial work [3] proposed the Flush Task (FT) mechanism to cleanse the state of shared resources to prevent the sensitive data on the shared resource from being gleaned by attackers. It also proposed a real-time scheduling algorithm and a schedulability analysis that take into account the timing overhead of the FT mechanism as a security constraint. There have been following studies designed for preemptive Fixed Priority (FP) and preemptive Dynamic Priority (DP) scheduling for enhancing the timing guarantees [2], [4].

While the above-mentioned studies have focused on single-criticality systems, nowadays RTSes are evolving into Mixed-Criticality Systems (MCSes) for which multiple components with different criticalities are integrated onto a single shared platform [7]. Such system characterizes modern real-time embedded systems (e.g., aerospace) in which a deadline miss by a high-criticality function can be disastrous (e.g., flight control), but losing a low-criticality function only moderately affects the quality of service (e.g., sensing rate), which requires a new principle of real-time scheduling effectively utilizing limited computing resources of embedded systems. Motivated by the necessity and following the up to date research trend in the real-time systems research community, we address the real-time scheduling problem of MCS in this paper.

Like single-criticality real-time systems, MCSes are also attractive targets for attackers aiming at obtaining sensitive data on the shared resources used by real-time tasks since their sub-systems are designed by different vendors and have different levels of criticality and security [3]. Taking an avionic system as an example, the sub-system that controls a camera to capture image or communicates the processed images back to the command center can be sourced from a more trustworthy vendor in terms of security. On the other hand, another sub-system calculating the flight path and controlling codes to manage the engine can be sourced from a more trustworthy vendor in terms of criticality. Since the former sub-system deals with more sensitive data in confidentiality, it can be placed at a higher security-level than the latter sub-system. In this case, the compromised sub-system having a lower security-level can glean the sensitive data on the shared resources while the two sub-systems are scheduled according to the given scheduler and actively communicate with each other.

Despite the structural vulnerability of the security of MCSes, most conventional MCS studies have focused on the real-time aspects of MCSes while security has received little attention [8], [9]. Motivated by this, we address the problem

Manuscript received November 4, 2016.
Manuscript revised May 8, 2017.
Manuscript publicized May 31, 2017.

[†]The authors are with Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea.
a) E-mail: Jinkyu.lee@skku.edu (Corresponding author)

DOI: 10.1587/transinf.2016EDP7447

of information leakage arising on the shared resources used by tasks with different security-levels of MCSes. We first define a new concept of a security constraint, and then we propose a new real-time scheduling algorithm for MCSes employing FT mechanism to incorporate the security constraint to mitigate information leakage on MCSes.

While a body of research has been made in the class of preemptive scheduling for MCS, those of non-preemptive scheduling for MCS have received little attention in spite of its necessity. In particular, non-preemptive scheduling is advantageous compared to preemptive scheduling in many practical systems whose preemption/migration overhead (e.g., interrupts and transactional operations) is prohibitively expensive [10]. Also, program locality can be destroyed by preemption, increasing runtime overhead due to cache miss and pre-fetch mechanisms, which make difficult to predict worst-case execution time [11], [12]. In control applications, non-preemptive scheduling simplifies control techniques for delay compensation at design time since every task has the same execution duration (interval between start and end times) [13]. In this paper, we focus on non-preemptive FP scheduling [3], and we adopt the AMC scheme [8] using a concept of mode change, which appears to be the best performing scheme among the MC scheduling schemes [14].

We also present a new schedulability analysis for the AMC scheme for which a security constraint is incorporated. We first derive the worst-case response time analysis for non-preemptive FP scheduling for AMC scheme based on the understanding how a mode change of AMC scheme influences executions of low- or high-criticality tasks. Then, we extend it to consider a security constraint, which employs two types of the interferences: (*a*) the worst-case interference from higher priority tasks and (*b*) from FTs invoked between them. The upper-bound of (*a*) depends on the timing of a mode change since low-criticality tasks cannot contribute to interference after a mode change. Thus we investigate how a security constraint influences the timing of mode change inducing the worst-case interference. The upper-bound of (*b*) depends on not only the timing of the mode change but also the ordering of security-levels of interfering tasks before and after the mode change. We discuss how to safely upper-bound the maximum number of FTs before and after a mode change independently, and then incorporate them into schedulability analysis.

The contributions of this paper are summarized as follows:

- We propose a new concept of security constraint to mitigate the information leakage arising on the shared resources used by real-time tasks with different security-levels in MCSes and propose a new MC scheduling policy incorporating it (Sect. 3).
- We derive response time analysis (RTA) [15] for non-preemptive FP for an MCS (Sect. 5.1).
- We introduce a new RTA providing for non-preemptive FP scheduler incorporating our proposed security constraint for MCSes (Sect. 5.2).
- We conduct experiments for comparison of our proposed mechanisms with conventional approaches, and we discuss how the variation of the parameters representing the features of an MCS and a security constraint affects the performance of each technique (Sect. 6).

Organization The MCS model considered in this paper is formally defined in Sect. 2. The existing scheduling scheme and security constraint for preventing information leakage are described in Sect. 3. The existing mechanism for the upper-bound of the maximum number of FTs is described in Sect. 4 as a background matter. A new schedulability analysis of non-preemptive FT for AMC scheme is described in Sect. 5. A new schedulability analysis considering a security constraint is proposed in Sect. 6. Performance of the proposed mechanism is evaluated in Sect. 7. Section 8 discusses the related work, and finally, Sect. 9 concludes with a discussion of future work.

2. Adversary and System Model

In this section, we present the adversary model describing the capability of attackers and the system model.

2.1 Adversary Model

We assume that an adversary can compromise some tasks or insert new tasks having low security-levels in an MCS to obtain sensitive data used by a task having a high security-level on the shared resource such as cache. For example, the adversary can launch a side-channel attack by compromising a lower security-level task to glean sensitive data on the shared cache used by higher security-level task. We also assume that the inserted task can consider real-time guarantee so that the task cannot be detected immediately. Adversaries that can tamper with the system operation are beyond the scope of this paper.

2.2 System Model

We consider a system that consists of K number of components, and each component has different criticality level X. Each component has the finite set of sporadic tasks. For simplicity, we assume a system comprising two criticality level HI and LO, which stand for the high- and low-criticality level, respectively. For a task set τ, we will characterize each MC (Mixed-Criticality) sporadic task $\tau_i \in \tau$ by a 6-tuple of the following parameters: $\tau_i = (X_i, C_i^{LO}, C_i^{HI}, D_i, T_i, S_i)$, where

- $X_i \in \{HI, LO\}$ denotes the criticality of the task. We assume that a task τ_i with $X_i = HI$, called a HI-criticality task, should be certified correctly by the Certification Authorities (CAs) while a task τ_i with $X_i = LO$, called a LO-criticality task, is not the target of the CAs.
- C_i^{LO} and C_i^{HI} are the worst-case execution times

(WCETs) for the high and low criticalities, respectively, and they satisfy $C_i^{HI} \geq C_i^{LO}$ since the high assurance of the high-criticality requires more conservative analysis for WCET.
- D_i denotes the relative deadline and T_i denotes the minimum separation between arrival times of two consecutive jobs. The n-th released job of a task τ_i is denoted by j_i^n; we will omit the superscript from j_i^n when no ambiguity arises.
- S_i denotes the security-level of a task τ_i. As a set of security-levels S forms total order, any pair of two tasks (τ_i, τ_j) exhibits two relationships: $S_i \prec S_j$ meaning τ_i has a higher security-level than τ_j, or $S_j \prec S_i$ otherwise.

We consider the constrained-deadline sporadic task system, meaning that $D_i \leq T_i$ for each task $\tau_i \in \tau$. We focus on non-preemptive FP, and assume quantum-based time. We consider an MCS with a uniprocessor.

Behaviors. For the different runs, a given task system shows two behaviors. *Criticality of a behavior* is defined as the minimum-criticality level that all released jobs do not actually execute for more than its WCET of the criticality level. For our two-level criticality model, if all released jobs j_i executed or are executing for less than its C_i^{LO}, by the definitions of *criticality of a behavior*, the current criticality level of behavior is low. On the other hand, if any job j_i executed or is executing for more than its C_i^{LO}, the current criticality level of behavior is high. For simplicity, we refer to the behavior of high- and low-criticality levels as HI-mode and LO-mode, respectively.

Correctness. The algorithm for scheduling MC task systems is said to be correct if and only if it satisfies the properties that

- In LO-mode, every job j_i of every task τ_i finishes its execution within D_i time units; and
- In HI-mode, every job j_i of every HI-criticality task finishes its execution within D_i time units.

For MCSes, a large body of fixed priority scheduling has been proposed such as CrMPO (Criticality Monotonic Priority Ordering) [14], SMC (Static Mixed-Criticality) [9], and AMC (Adaptive Mixed-Criticality) [8]. In this paper, we focus on AMC scheme since it appears to show the best performance among them and there a lot of further studies for MCSes have been based on it [16]–[18].

3. AMC Scheme Incorporating Security Constraints

In this section, we first review the conventional AMC scheme, and then we introduce a new AMC scheme incorporating a security constraint proposed to mitigate the problem of information leakage arising on the shared resources.

3.1 Existing AMC Scheme

In the AMC scheme, all LO-criticality jobs are dropped when any job j_i executes for more than its LO-criticality WCET C_i^{LO}. The ability to stop any job executing for more than its C_i^{LO} is based on the platform support that can monitor the execution of the job. This is an ability to measure the duration of the execution of the target job [8].

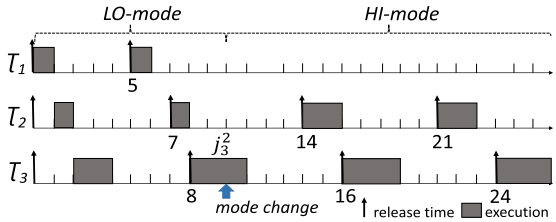


Fig. 1 Example of AMC scheme taking a mode change.

Table 1 Task parameters.

τ_i	X_i	C_i^{LO}	C_i^{HI}	D_i	T_i
τ_1	LO	1	-	5	5
τ_2	HI	1	2	7	7
τ_3	HI	2	3	8	8

Figure 1 illustrates an AMC scheme using runtime monitoring with RM priority assignment for a task set with the parameters described in Table 1.

As seen in Fig. 1, a job of τ_1 having the shortest period executes with the highest priority, and then jobs of τ_2 and τ_3 execute according to rate monotonic (RM) priority assignment scheme. We assume that a mode change occurs by the second released job j_3^2 of the task τ_3 at the time 10. Before the mode change, the execution of each job is completed within its C_i^{LO}. At time 10, a mode change occurs, and then a LO-criticality task τ_1 is dropped out and the two HI-criticality tasks τ_2 and τ_3 are assumed to complete its execution within its C_i^{HI}

3.2 New AMC Scheme Incorporating Security Constraints

Many mechanisms to mitigate or completely prevent information leakage have been proposed [19]–[21]. One of the mechanisms is to partition cache with hardware or software approaches [21]. However, such approaches are dependent on the type of the shared resources and can negatively influence the execution time of task since each task could use the limited shared resource. Instead, some studies used a general flushing mechanism [2]–[4]. Whenever there is a possibility of information leakage, a function to clear the state of the shared resource is invoked, which is called FT. The cache, for example, can be flushed and a row of DRAM can be closed when a task executes immediately after the execution of another task since the newly executing task could inspect the contents of the shared resource that were recently used.

As we assume real-time tasks having different security-levels meaning different levels of security requirements, two directions of information leakage are possible: from the task of the higher security-level τ_H to the task of the lower security-level τ_L and vice versa, for any two consecutively

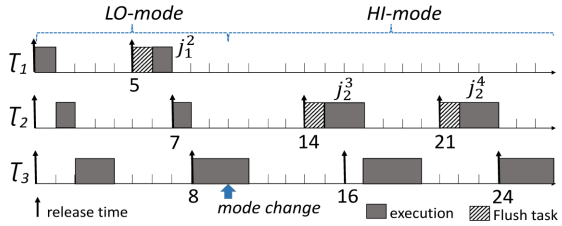


Fig. 2 Example of AMC scheme incorporating a security constraint.

executing tasks. Note that we assume the total order of security-levels meaning that all tasks have different security-levels. In this paper, likewise existing studies [3], [4], we focus on the information leakage arising from τ_H to τ_L since the information leakage from τ_L to τ_H is not as critical as that from τ_H to τ_L and considering two directions of information leakage causes more FTs, which often results in task sets becoming unschedulable. Thus, we incorporate the following security constraint into real-time scheduling to address the information leakage problem; *for any two consecutively executing tasks under any real-time scheduler, a FT to cleanse the state of shared resources should be invoked between the two tasks if the former is τ_H and the latter is τ_L.* We assume that a FT is executed together with a scheduled job j_i if j_i follows a job j_j, with $S_j \prec S_i$. Hence, a FT can increase the execution time of j_i by C_{ft}, and it also executes non-preemptively before the execution of j_i.

Figure 2 illustrates an example of scheduling applying a security constraint for tasks with parameters in Table 1. Additionally, we assume that tasks τ_1, τ_2 and τ_3 have their security ordering as $S_3 \prec S_2 \prec S_1$. Comparing to the example in Fig. 1, three FTs are invoked at time 5, 14 and 21 to prevent information leakage between a transition $\tau_H \rightarrow \tau_L$; in this case, $\tau_3 \rightarrow \tau_1$, $\tau_3 \rightarrow \tau_2$ and $\tau_3 \rightarrow \tau_2$. As a FT executes non-preemptively with a priority higher than all of the real-time tasks, the execution of the jobs j_1^2, j_2^3 and j_2^4 are delayed by the amount of execution time of the FT (1 time unit in this case). Comparing the two cases illustrated by Figs. 1 and 2 respectively, a job j_i of the former case (Fig. 1) can be interfered by the higher priority jobs only while a job j_i of the latter case (Fig. 2) can be interfered by both of the higher priority jobs and FTs invoked between the release and the completion of j_i. Thus, to test the schedulability of τ_i (i.e., test whether every job of τ_i completes its execution within its relative deadline D_i), the upper-bound of the worst-case interference from higher priority jobs than j_i of the task τ_i and the maximum number of FTs invoked between them is crucial.

In the following section, we introduce a conventional mechanism upper-bounding the maximum number of FTs interfering with a job j_i, which will be embedded into our new RTA to analyze the schedulability of the AMC scheme incorporating a security constraint.

4. Existing Mechanism for FT Bounds

In this section, we review the mechanism upper-bounding the maximum number of FTs invoked between the interfering jobs of a job j_i of a task τ_i, which is used to test the schedulability of τ_i introduced in [3].

The exact number of FTs invoked between the interfering jobs of j_i is derived by three factors of interfering jobs: (*a*) the ordering of the security level S, (*b*) the number of the interfering jobs of a task τ_j denoted by N_j and (*c*) the sequence of the executions of the interfering jobs induced by the actual arrival time of each interfering job. However, the mechanism proposed in [3] only depends on the ordering of the security-level S and the number of jobs N_j interfering with τ_i but not the actual arrival time of each interfering job. Thus, it does not derive the exact number of FTs but a safe upper-bound of the maximum number of FTs invoked between interfering jobs of j_i. Intuitively, it finds a job sequence inducing the maximum number of FTs invoked between interfering jobs among possible permutations of job sequences. The function to derive the maximum number of FTs is denoted by $N_{ft}(S, \{N_j \,|\tau_j \in hep(i)\})$ where $hep(i)$ is a set of tasks having higher or equal priority. To derive the value of $N_{ft}(S, \{N_j \,|\tau_j \in hep(i)\})$, the following two definitions are used.

Definition 1 (Valid job sequence [3]). A valid job sequence ψ is a sequence of $\sum_{\tau_j \in hep(i)} N_j + 2$ jobs. The first job can be any job; the last job should be a job of τ_i; and intermediate jobs form any permutation of the union of N_j jobs for each task τ_j in $hep(i)$. Let $\Psi(S, \{N_j \,|\tau_j \in hep(i)\})$ be the set of valid job sequences for S and $\{N_j|\tau_j \in hep(i)\}$.

Definition 2 (Number of FTs for ψ [3]). Let $N(\psi)$ be the number of FTs required in the valid job sequence ignoring the arrival times of interfering jobs. For example, for any consecutively executing jobs j_a and j_b in a valid job sequence, an FT is invoked if $S_a \prec S_b$.

Although some valid job sequences do not represent a valid schedule (note that we do not make any assumption of the arrival time of the interfering jobs), a valid job sequence containing the maximum number of FTs safely upper-bounds the actual number of FTs. Thus, we can upper-bound the maximum number of FTs invoked between interfering jobs as follows:

$$N_{ft}(S, \{N_j|\tau_j \in hep(i)\}) = \max_{\psi \in \Psi(S, \{N_j|\tau_j \in hep(i)\})} N(\psi). \quad (1)$$

Enumerating all of the possible permutations of the valid job sequences for the upper-bound of the maximum number of FTs requires factorial time.

In [3] a mechanism showing the polynomial time complexity ($O(|V|^3)$) is proposed through transformation of the problem of maximization of the number of FTs into a max flow problem, where $|V|$ is the number of the interfering jobs of j_i under test.

Each of the jobs in a valid sequence is mapped onto the pair of *sender* and *receiver* nodes. An edge with capacity 1 is then added between a sender node of a higher

security-level and a receiver node of a lower security-level as it represents the possibility of a FT invocation between two consecutively executing jobs in a valid sequence. In detail, the graph for maximizing the number of FT invocations in a valid sequence is constructed as follow:

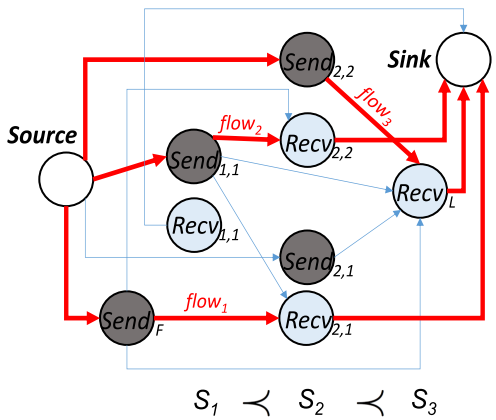
Definition 3 (FT graph [3]). The FT Graph for S and $\{N_j|\tau_j \in hep(i)\}$ is a flow graph (V, E). A set of vertexes V contains the following vertexes:

1. a source vertex and a sink vertex;
2. a sender vertex $Send_F$ representing a blocking job and a receiver vertex $Recv_L$ representing a job under test;
3. for each $\tau_j \in hep(i)$ N_j sender vertexes $Send_{j,1}, \cdots, Send_{j,N_j}$ and N_j receiver vertexes $Recv_{j,1}, \cdots, Recv_{j,N_j}$, which represent N_j interfering jobs of the task τ_j.

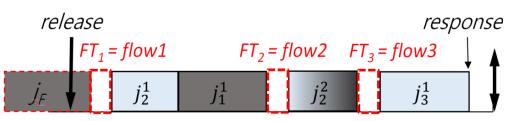
A set of directed edges E, where all edges have a capacity of 1 contains the following edges:

1. an edge from the source to every sender vertex;
2. an edge from every receiver vertex to the sink;
3. for each sender vertex $Send_x$ except a source vertex, an edge from the sender vertex $Send_x$ to any receiver vertex $Recv_y$ except a sink vertex, if the job j_x represented by $Send_x$ and the job represented by $Recv_y$ have security-levels such that $S_x \prec S_y$.

Figure 3 illustrates an example of a FT graph deriving a max flow (Fig. 3(a)) and a corresponding valid sequence (Fig. 3(b)) for a task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ and a security ordering, $S_1 \prec S_2 \prec S_3$. Let τ_3 be the task under test and $N_1 = 1, N_2 = 2$. Thus, we consider the following jobs: j_F, j_1^1, j_2^1, j_2^2 and j_3^1 where j_F can be any job having a lower



(a) Example of FT graph representing a max flow



(b) Corresponding valid sequence

Fig. 3 Example of FT graph and corresponding valid sequence.

priority than j_3^1. Note that j_F creates sender nodes $Send_F$ only, since j_F should be the first job in the valid sequence. On the other hand, j_3^1 creates a receiver node $Recv_L$ only, since it cannot be ahead of any FT; it should be the end of the valid sequence as it is the job under test. Each edge has a capacity of 1, and bold red lines represent flows. Each flow path starts from a source node and ends with a sink node. A flow between two intermediate nodes, a sender node and a receiver node, represents a FT invocation in the valid sequence. Therefore, the max flow in Fig. 3(a) produces a corresponding valid sequence (Fig. 3(b)) containing three FTs. Note that the resultant valid sequence does not always present a valid schedule but it safely upper-bounds the maximum number of FT invocations.

Lemma 4.1. (from [3]) *For the given max flow value F' derived by the FT graph (V, E) for S, the maximum number of FTs in the valid sequence is upper-bounded by F'.*

5. New Schedulability Analysis for AMC

In this section,we propose a new RTA for non-preemptive FP scheduling for AMC scheme. Before we develop a new RTA technique, we revisit the existing RTA for non-preemptive FP scheduling for the single-criticality system introduced in [3]. The worst-case response time R_i for non-preemptive FP scheduling is upper-bounded by the following recursive form.

$$R_i = B_i + C_i + \sum_{\tau_j \in hep(i)} N_j \cdot C_j, \tag{2}$$

where B_i is the maximum blocking time induced by a lower priority task calculated by

$$B_i = \max_{\tau_k \in lp(i)} C_k - 1, \tag{3}$$

where $lp(i)$ is a set of tasks having lower priority, and N_j is the number of interfering jobs of higher priority task τ_j obtained by

$$N_j = \left\lfloor \frac{R_i - C_i}{T_j} + 1 \right\rfloor. \tag{4}$$

Unlike preemptive scheduling, a task τ_i can be interfered by a lower priority task τ_k if τ_k is released before the release of τ_i. Since the lower priority task having the largest WCET safely upper-bound the maximum blocking time B_i for the worst-case interference, B_i is derived by Eq. (3). N_j (in Eq. (4)) is calculated with the similar mechanism for RTA for preemptive FT scheduling [15] but RTA for non-preemptive FP scheduling has two different terms: (1) "$-C_i$" term, and (2) "+1" and floor terms. Figure 4 illustrates the two worst-case scenarios in which a job of τ_i is interfered by τ_j as much as possible. Let t be the time instant when a job of τ_i is released. In the first worst-case scenario (Fig. 4 (a)), only jobs of τ_j released before $t + R_i - C_i$ can interfere

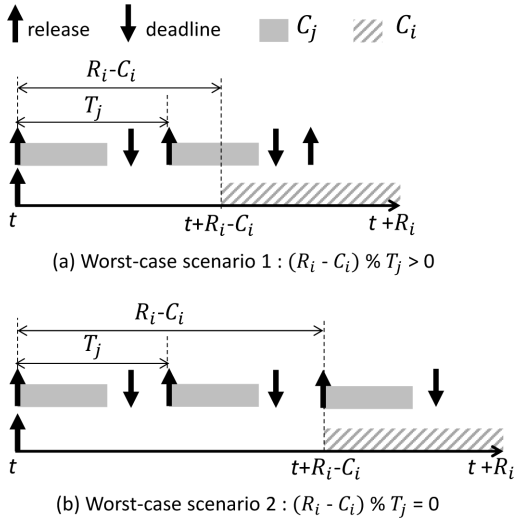


Fig. 4 Two worst-case scenarios in which a job of τ_i is maximally interfered by higher priority jobs in each iteration of RTA for non-preemptive FP scheduling.

with the job of τ_i since jobs execute non-preemptively. Figures 4 (b) presents the special case in which a job of τ_j is released at $t+R_i-C_i$. In this case, the job released at $t+R_i-C_i$ also can interfere with the job of τ_j. "+1" and floor terms in Eq. (4) are used to cover both scenarios. Note that RTA for preemptive FP scheduling uses ceil term without "+1" term but it cannot cover the second worst-case scenario; actually, the second scenario does not happen in preemptive FP scheduling. Once RTA computes the worst-case response time R_i of each task $\tau_i \in \tau$, we can check the schedulability of τ by comparing R_i with the relative deadline of each task (*i.e.* tests if $R_i \leq D_i$).

The schedulability analysis of AMC scheme consists of three phases [8]: schedulability of LO-mode, HI-mode and mode change. First two phases consider the stable mode where a mode change does not occur, and the third one considers, for HI-task, the dynamic mode starting at the LO-mode, going through a mode change and finished at the HI-mode. To test schedulability of a LO-criticality task, RTA for the LO-mode is applied since LO-criticality tasks execute only in the LO-mode (note that all of the LO-criticality tasks are dropped out in the HI-mode according to the policy of the AMC scheme). On the other hand, HI-criticality tasks can execute only in the LO- or HI-mode, or can experience a mode change. We derive the three worst-case response times of a HI-criticality task for the LO-mode, HI-mode and mode change respectively. Then, we show that the worst-case response time of a HI-criticality task for a mode change is always safer than that of LO-mode and HI-mode in this section. Therefore, we derive the following three types of worst-case response time of each task; R_i^{LO} and R_i^{HI} for two stable modes, and R_i^{TR} for a dynamic mode taking into account a mode change.

- R_i^{LO}: the worst-case response time of a LO-criticality task or a HI-criticality task executing only in LO-mode.
- R_i^{HI}: the worst-case response time of a HI-criticality task executing only in HI-mode.
- R_i^{TR}: the worst-case response time of a HI-criticality task whose execution starts in LO-mode and ends in HI-mode, which experiences a mode change.

Since all of the jobs of HI- or LO-criticality tasks including a job under test and the interfering jobs complete its execution within its C_i^{LO} in the LO-mode, only execution times of the LO-mode are used for the analysis. On the other hand, since only jobs of the HI-criticality tasks execute for C_i^{HI} in the worst-case in HI-mode, only execution times of HI-mode are used for the analysis. Thus, the worst-case response time R_i^{LO} for the LO-mode and R_i^{HI} for the HI-mode are directly derived from Eq. (2) consisting of the summation of the maximum blocking time, the WCET of τ_i under analysis and the worst-case interference from the higher priority tasks as follow:

$$R_i^M = B_i^M + C_i^M + \begin{cases} \sum_{\tau_j \in hep(i)} N_j^M \cdot C_j^M \\ \qquad \text{if } M = LO, \\ \sum_{\tau_j \in hepH(i)} N_j^M \cdot C_j^M \\ \qquad \text{otherwise,} \end{cases} \tag{5}$$

where $M \in \{LO, HI\}$ denotes the target mode for the analysis, $hepH(i)$ is a set of HI-criticality tasks having higher or equal priority, B_i^M is the maximum blocking time in the LO- or HI-mode computed as

$$B_i^M = \begin{cases} \max_{\tau_k \in lp(i)} C_k^M - 1 \\ \qquad \text{if } M = LO, \\ \max_{\tau_k \in lpH(i)} C_k^M - 1 \\ \qquad \text{otherwise,} \end{cases} \tag{6}$$

where $lpH(i)$ is a set of HI-criticality tasks having lower priority and N_j^M is the number of interfering jobs of τ_i calculated by

$$N_j^M = \left\lfloor \frac{R_i^M - C_i^M}{T_j} + 1 \right\rfloor. \tag{7}$$

Note that $hepH(i)$ is used for the analysis of the HI-mode since only HI-criticality tasks execute and can interfere with τ_i in the HI-mode.

Since a critical instant[†] of the AMC scheme is not clear as it is impossible to know the exact timing of a mode change in advance, to derive the exact response time analysis for the schedulability analysis considering a mode change is naturally intractable. Thus, we consider the sufficient analysis [8] for schedulability for AMC scheme considering a mode change.

The equation of the standard RTA for non-preemptive scheduling is formed by the summation of the blocking time, the WCET of a task under analysis and the worst-case interference from higher priority tasks. In the RTA for the AMC scheme, the interference from higher priority of HI-criticality tasks and that of LO-criticality tasks are

[†]A critical instant is defined as a time instant inducing the worst-case response time [22].

independently considered since LO-criticality tasks do not contribute to the interference after a mode change. Thus, the worst-case response time of HI-criticality task taking a mode change is derived by the following combination of forms

$$R_i^{TR} = B_i^{TR} + C_i^{HI} + I_L + I_H, \tag{8}$$

where I_L is the worst-case interference from LO-criticality tasks, I_H is the worst-case interference from HI-criticality tasks, and B_i^{TR} is the maximum blocking time calculated by

$$B_i^{TR} = \max(\max_{\tau_k \in lpL(i)} C_k^{LO}, \max_{\tau_k \in lpH(i)} C_k^{HI}) - 1, \tag{9}$$

where $lpL(i)$ is a set of LO-criticality tasks having lower priority than τ_i. As we consider non-preemptive scheduling, a job j_i under test can be blocked by a lower priority job either of LO- or HI-criticality [23]. If LO-criticality, the job executes for less than its C_i^{LO}; otherwise, it can execute for more than C_i^{LO} since a mode change can occur during the execution of the blocking job. Thus, we upper-bound the maximum blocking time induced by a lower priority job according to the criticalities of the blocking jobs as seen in Eq. (9).

Since all LO-criticality tasks are dropped out after a mode change by the policy of the AMC scheme, LO-criticality tasks having higher priority than τ_i can interfere until the mode change. Since a mode change to HI-mode occurs before R_i^{LO}, the number of interfering jobs of the LO-criticality is maximized when the mode change occurs at R_i^{LO}. Based on the reasoning, I_L is upper-bounded by

$$I_L = \sum_{\tau_j \in hepL(i)} \left\lfloor \frac{R_i^{LO} - C_i^{LO}}{T_j} + 1 \right\rfloor C_j^{LO}, \tag{10}$$

where $hepL(i)$ is the set of LO-criticality jobs having higher or equal priority. For the safe upper-bound of I_H, we assume that a mode change occurs at R_i^{LO} to maximize the number of HI-criticality interfering jobs, and all released HI-criticality interfering jobs executes for C_i^{HI} to maximize the amount of interference. Thus, I_H is safely upper-bounded by

$$I_H = \sum_{\tau_j \in hepH(i)} \left\lfloor \frac{R_i^{TR} - C_i^{HI}}{T_j} + 1 \right\rfloor C_j^{HI}. \tag{11}$$

Based on the above reasoning regarding the upper-bound of the worst-case response time of a HI-criticality task for a mode change, we derive the following lemma.

Lemma 5.1. *The worst-case response time R_i^{TR} of a HI-criticality task τ_i for a mode change is upper-bounded by*

$$R_i^{TR} = B_i^{TR} + C_i^{HI} + I_L + I_H. \tag{12}$$

Using the derived worst-case response times for the LO-mode, HI-mode and a mode change, the schedulability is tested by the following theorem.

Theorem 5.2. *A task set τ is schedulable by AMC scheme if for each LO-criticality task τ_i satisfies*

$$R_i^{LO} \le D_i, \tag{13}$$

and for each HI-criticality task τ_i satisfies

$$R_i^{TR} \le D_i. \tag{14}$$

Proof. LO-criticality tasks execute only in LO-mode. Thus, R_i^{LO} sufficiently upper-bounds the worst-case response time of every LO-criticality task. A HI-criticality task can execute only in LO-mode or HI-mode, and can experience a mode change. For a given HI-criticality task τ_i, R_i^{TR} is always greater than R_i^{LO} and R_i^{HI}. Since R_i^{TR} is safely upper-bounded by Lemma 5.1, the worst-case response time of a HI-criticality task is safely upper-bounded by R_i^{TR} regardless of a mode change occurrence. □

6. New Schedulability Analysis Embedding FT Bounds

In this section, we propose a new schedulability analysis approach for non-preemptive FP for AMC scheme employing FT mechanism as a security constraint (proposed in Sect. 3.2) by combining our proposed RTA for non-preemptive FP for AMC scheme (proposed in Sect. 5) and the previous mechanism upper-bounding the maximum number of FTs invoked in the valid sequence (illustrated in Sect. 4).

We first consider the worst-case response times for LO-mode and HI-mode, and then we discuss how to upper-bound it for a mode change. For a mode change, we first identify the worst-case situations where the maximum number of FTs invoked before and after a mode change separately. Then we upper-bound the maximum number of FTs of each situation. Based on the same reasoning mentioned in Sect. 5, we derive the following three types of the worst-case response time to test the schedulability of LO- and HI-criticality tasks.

- R_i^{LO+}: the worst-case response time considering the upper-bounded interference of FT of a LO-criticality task or a HI-criticality task executing only in the LO-mode.
- R_i^{HI+}: the worst-case response time considering the upper-bounded interference of FT of a HI-criticality task executing only in the HI-mode.
- R_i^{TR+}: the worst-case response time considering the upper-bounded interference of FT of a HI-criticality task whose execution starts in the LO-mode and ends in the HI-mode, which takes a mode change.

Note that we use "+" notation if the value of the term increases due to the consideration of a security constraint (*i.e.* the worst-case response time R_i^+ or the maximum blocking time from a lower priority task B_i^+, which consider a security constraint).

As illustrated in the previous section, RTA derives the worst-case response time for LO-mode and HI-mode by the

summation of three terms: the maximum blocking time induced by a lower priority task, the WCET of τ_i, and the worst-case interference of higher priority tasks. By considering a security constraint, a job j_i of τ_i can be additionally interfered by FTs invoked between interfering jobs of j_i. As the four factors are calculated independently, the worst-case response time taking into account the maximum number of FTs invoked between interfering jobs is directly extended from Eq. (5), which is the summation of the four terms as follow:

$$R_i^{M+} = B_i^{M+} + C_i^M + \left\{ \begin{array}{ll} \sum_{\tau_j \in hep(i)} N_j^{M+} \cdot C_j^{M+} & \\ & \text{if } M = LO \\ \sum_{\tau_j \in hepH(i)} N_j^{M+} \cdot C_j^{M+} & \\ & \text{otherwise.} \end{array} \right\} + N_{ft}(S, \{N_j^{M+} | \tau_j \in hep(i)\}) \cdot C_{ft}, \quad (15)$$

where B_i^{M+} is the maximum blocking time in the LO- or HI-mode computed as

$$B_i^{M+} = \left\{ \begin{array}{ll} \max_{\tau_k \in lp(i)} C_k^{M+} - 1 & \\ & \text{if } M = LO, \\ \max_{\tau_k \in lpH(i)} C_k^{M+} - 1 & \\ & \text{otherwise.} \end{array} \right. \quad (16)$$

$C_k^{M+} = C_k^M + C_{ft}$ if there exits any task having lower priority and higher security-level than any task $\tau_k \in lp(i)$; $C_k^{M+} = C_k^M$, otherwise. In other words, if any task having higher security-level executed just before $\tau_k \in lp(i)$ so that a FT should be invoked between the consecutive two tasks, then we add C_{ft} to the blocking time. N_j^{M+} is the number of jobs of τ_j interfering with τ_i calculated by

$$N_j^{M+} = \left\lfloor \frac{R_i^{M+} - C_i^M}{T_j} + 1 \right\rfloor. \quad (17)$$

Now, we consider the worst-case response time taking into account the maximum number of FTs for a mode change. Unlike the case of single criticality scheduling, the situations before and after a mode change should be considered separately since LO-criticality jobs cannot contribute to the invocations of FTs after a mode change. Therefore, we derive the maximum number of FTs before and after a mode change independently, and then we derive a safe upper-bound of the worst-case response time for a mode change.

Before we upper-bound the maximum number of FTs, we introduce the following lemma.

Lemma 6.1. *Let N_a and N_b be the number of elements of a set of J_A and J_B, respectively. If a set of job J_A includes a set of job J_B, then it satisfies the following*

$$N_{ft}(S, \{N_b | J_B\}) \leq N_{ft}(S, \{N_a | J_A\}). \quad (18)$$

Proof. We proof it by induction. Suppose J_B has a job j_1 only and J_A has two jobs j_1 and j_2. The function value of J_B should be 0 and that of J_A should be 0 or 1 since a single job cannot induce any FT and two jobs can induce a single FT at most. Thus the inequality is satisfied. Suppose J_B has N jobs, and J_A include all of the jobs in J_B and another job j_i. j_i does not change the configuration (edges and nodes) of the FT graph of J_B, but just add its own edges and nodes, which means the FT graph of J_A also includes that of J_B. Thus the inequality is also satisfied. Therefore, the lemma holds. □

Let N_j^b be the number of interfering jobs of task τ_j released before a mode change when the mode change occurs at R_i^{LO+}. N_j^b is directly calculated using Eq. (17) as follows:

$$N_j^b = \left\lfloor \frac{R_i^{LO+} - C_i^{LO}}{T_j} + 1 \right\rfloor. \quad (19)$$

We derive the following lemma to upper-bound I_{ft}^b with N_j^b.

Lemma 6.2. *The worst-case interference from the FT invocations before a mode change I_{ft}^b is upper-bounded by*

$$I_{ft}^b = N_{ft}(S, \{N_j^b | \tau_j \in hep(i)\}) \cdot C_{ft}. \quad (20)$$

Proof. Let J^b be the sets of interfering jobs executing before a mode change. As a mode change is delayed, the number of interfering jobs executing before a mode change in J^b monotonically increases. During this increase, J^b adds newly released jobs while conserving the existing jobs in J^b. By Lemma 6.1, the function value of J^b does not decrease while J^b adds newly released jobs. Therefore, I_{ft}^b is maximized when the mode change occurs at R_i^{LO+} since R_i^{LO+} is the latest time that the mode change can occur. If a mode change occurs at R_i^{LO+}, then it means that the mode change occurs by a job j_i under test and all of the released interfering jobs before a mode change are already executed. Thus, all of the jobs released before R_i^{LO+} participate in the function $N_{ft}(\cdot)$. By Lemma 4.1, it is safely upper-bounded. □

Let I_{ft}^a be the worst-case interference from FT invocations after a mode change, and J^a is a set of jobs executing after a mode change and contributing to I_{ft}^a. From Lemma 6.2, I_{ft}^b is maximized when the mode change occurs at R_i^{LO+} meaning a mode change is maximally delayed. However, it is not guaranteed that I_{ft}^a is maximized when the mode change is maximally delayed, which means that two worst-cases upper-bounding I_{ft}^b and I_{ft}^a cannot occur simultaneously. Let us consider the following two cases for a job j_i under test.

1. A mode change occurs before R_i^{LO+} and a large number of higher priority HI-criticality jobs have been released but not executed yet. Those jobs will interfere with j_i after the mode change and contribute to I_{ft}^a.
2. A mode change occurs at R_i^{LO+} and all of the jobs released before R_i^{LO+} finish its executions. J^a only includes the job released after the mode change.

Comparing the above two cases, I_{ft}^a for the first case

can be greater than that for the second case. Thus, the mode change at R_i^{LO+} maximizes I_{ft}^b but not I_{ft}^a. Based on the reasoning, we use the RTA for the HI-mode deriving R_i^{HI+} for the safe upper-bound of I_{ft}^a, since it safely upper-bounds the maximum number of the interfering jobs between interfering jobs in HI-mode (see Eq. (15)). Therefore, for the safe upper-bound of I_{ft}^a, we derive the following lemma.

Lemma 6.3. *The worst-case interference from the FT invocations after a mode change I_{ft}^a is upper-bounded by*

$$I_{ft}^b = N_{ft}(S, \{N_j^a | \tau_j \in hepH(i)\}) \cdot C_{ft}, \tag{21}$$

where, N_j^a is the number of interfering jobs in the HI-mode computed by

$$N_j^a = \left\lfloor \frac{R_i^{HI+} - C_i^{HI}}{T_j} + 1 \right\rfloor. \tag{22}$$

Proof. Equation (15) safely upper-bounds the maximum number of the interfering jobs and the maximum number of the FTs invoked between interfering jobs in the HI-mode. With the same reasoning as Lemma 6.2 using Lemmas 6.1 and 4.1, this lemma holds. □

Based on the reasoning, we derive the following lemma.

Lemma 6.4. *The worst-case response time taking into account FTs for a mode change is upper-bounded by the following combination of terms:*

$$R_i^{TR+} = B_i^{TR+} + C_i^{HI} + I_L + I_H + I_{ft}^b + I_{ft}^a, \tag{23}$$

where B_i^{TR+} is the maximum blocking time calculated by

$$B_i^{TR+} = \max(\max_{\tau_k \in lpL(i)} C_k^{LO+}, \max_{\tau_k \in lpH(i)} C_k^{HI+}) - 1. \tag{24}$$

Proof. The worst-case response time for a mode change is safely upper-bounded by Lemma 5.1 and, the worst-case interference from the FT invocations before a mode change I_{ft}^b and after a mode change I_{ft}^a are safely upper-bounded by Lemmas 6.2 and 6.3 respectively. Thus the lemma holds. □

Based on the derived worst-case response times for the LO-mode, HI-mode, and a mode change, we present the final RTA as follows:

Theorem 6.5. *A task set τ is schedulable by the AMC scheme incorporating a security constraint if for each LO-criticality task τ_i satisfies*

$$R_i^{LO+} \leq D_i, \tag{25}$$

and for each HI-criticality task τ_i satisfies

$$R_i^{TR+} \leq D_i. \tag{26}$$

Proof. R_i^{LO+} sufficiently upper-bounds the worst-case response time of a LO-criticality task since LO-criticality tasks execute only in the LO-mode. R_i^{TR+} is always greater than R_i^{LO+}, and R_i^{HI+} for any HI-criticality task. Since R_i^{TR+} is safely upper-bounded by Lemma 6.4, the worst-case response time of a HI-criticality task is safely upper-bounded by R_i^{TR+} regardless of a mode change occurrence. □

7. Evaluation and Discussion

In this section, we evaluate our approaches introduced in the previous sections. We first describe the simulation environment, including the task generation method. Then, we evaluate the performance of our proposed techniques.

7.1 Task Parameter Generation

We followed the task set generation method described in [3]. We set 10 base utilization† groups $[0.02+0.1\cdot i, 0.08+0.1\cdot i]$ for $i = 0, \cdots, 9$, and generated 2000 instances per group. For example, the first group contains task sets that the value of utilization of each task set is from 0.02 to 0.08. As we consider an MCS, we added two additional parameters: the criticality factor (CF) and the criticality mix (CM) introduced in [8]. CF denotes the fixed multiplier of the LO-criticality execution time producing HI-criticality execution time (*i.e.* $C_i^{HI} = CF \cdot C_i^{LO}$). CM denotes the probability to generate HI-criticality task (*i.e.* $0 \leq CM \leq 1$). Table 2 describes the task parameters used in our experiments.

7.2 Experiments

We investigated the performance of the following six schedulability analysis techniques for rate monotonic priority ordering scheme (first four techniques for non-preemptive scheduling and the others are for preemptive scheduling).

- UB-H&L: Task sets pass this test if they are schedulable both for the schedulability tests for non-preemptive FP scheduling of the LO- and HI-mode. It represents the upper-bounded performances of the rest of the considered techniques for non-preemptive FP scheduling, because it does not check the schedulability with a mode change.

Table 2 Task parameters for experiments.

Number of tasks	$N \in \{3, 4, \cdots, 10\}$
Task period	$p_i \in \{50, 100, \cdots, 1000\}$
Task execution time	$e_i \in \{5, 6, \cdots, 50\}$
Security-level of task	$S_i \in \{3, 4, \cdots, 10\}$
FT overhead	$C_{ft} \in \{1, 2, \cdots, 10\}$
Criticality Factor	$CF \in \{1, 1.5, \cdots, 5\}$
Criticality Mix	$CM \in \{0.1, 0.2, \cdots, 1\}$

†Utilization of a task τ_i is defined as C_i^{LO}/T_i for a LO-criticality task and C_i^{HI}/T_i for a HI-criticality task. Utilization of a task set τ is defined as the summation of utilizations of tasks in τ.

- AMC-non: the schedulability analysis for non-preemptive FP scheduling for the AMC scheme introduced in Sect. 5 not incorporating a security constraint.
- AMC-mf: the approach described in Sect. 6 considering a security constraint.
- AMC-ob: the schedulability analysis for non-preemptive FP scheduling for the AMC scheme considering a security constraint naively upper-bounding the maximum number of FTs invoked between interfering jobs. The maximum number of N_{ft} is calculated by $N_{ft} = N_{hep(i)} + 1$, where $N_{hep(i)}$ is the number of higher or equal priority jobs for each task τ_i.
- AMC-p: the schedulability analysis for preemptive FP scheduling for the AMC scheme proposed in [8].
- AMC-ob-p: the schedulability analysis for preemptive FP scheduling for the AMC scheme (based on [8]) considering a security constraint naively upper-bounding the maximum number of FTs invoked between interfering jobs. The maximum number of N_{ft} is calculated by $N_{ft} = 2 \cdot N_{hep(i)} + 1$.

Note that AMC-non, AMC-mf and AMC-ob share the same schedulability analysis to upper-bound the worst-case interference from the higher priority tasks for the LO-mode, HI-mode and mode change. However, AMC-non does not consider the worst-case interference from the FTs while AMC-mf and AMC-ob upper-bound it with their own approaches; AMC-ob considers that a FT is invoked in every transition of any two tasks $\tau_i \rightarrow \tau_j$ while AMC-mf uses a more sophisticated approach upper-bounding the maximum number of FTs by transforming it into a max-flow problem. Therefore, AMC-non and AMC-ob provide upper- and lower-bounds of performance of AMC-mf respectively. When it comes to AMC-p and AMC-ob-p, AMC-p does not consider a security constraint, and AMC-ob-p considers that a FT is invoked in every transition of any two tasks $\tau_i \rightarrow \tau_j$. The bounded number of FT of AMC-ob and AMC-ob-p ($N_{hep(i)} + 1$ and $2 \cdot N_{hep(i)} + 1$ respectively) are derived by the maximum number preemptions that can occur from each job; a job can occur preemption at most once and twice under any non-preemptive and preemptive scheduling respectively.

We first investigate how performances of schedulability analysis techniques for non-preemptive scheduling are varied with changing values of base utilization groups, CM and CF. Figure 5 plots the percentage of the task sets deemed schedulable for 2000 tasks in each base utilization group with $C_{ft} = 5$, $CF = 2$ and $CM = 0.5$. As seen in Fig. 5, UB-H&L provides an upper-bound of performance, and the rest of the three techniques produce three distinctive results. As AMC-non does not consider a security constraint, it performs better than AMC-mf and AMC-ob since the executions of tasks are not hindered by the FTs. AMC-ob performs badly because of its naive mechanism (*i.e.* $N_{ft} = N_{hep(i)} + 1$) for the upper-bound of the maximum number of FT invocations, while AMC-mf outperforms AMC-ob thanks to tight calculation of the interference/blocking. As seen in Fig. 5, a large portion of the tasks sets of AMC-mf is still schedulable compared to AMC-non even when a security constraint is incorporated.

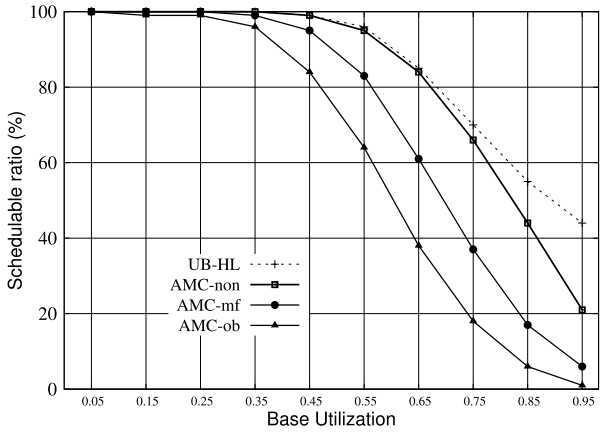


Fig. 5 Percentage of schedulable task sets ($C_{ft} = 5, CF = 2, CM = 0.5$).

In the following evaluations, we use the weighted schedulability measure $W_y(p)$ [24] for schedulability analysis y as a function of parameter p. Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for task set τ with the parameter value p, and then $W_y(p)$ is calculated as follows:

$$W_y(p) = \sum_{\forall \tau} u(\tau) \cdot S_y(\tau, p) / \sum_{\forall \tau} u(\tau), \tag{27}$$

where $u(\tau)$ is the utilization of τ. The weighted schedulability measure rates high value to the task set having high utilization deemed schedulable, and it reduces 3-dimensional result to 2-dimensional result.

Figures 6 and 7 show how the experimental results are changed by the varying value of CF and CM respectively; the value of CM for Fig. 6 is set to 0.5, and that of CF for Fig. 7 is set to 2.0. As seen in Figs. 6 and 7, the weighted schedulability of each technique decreases as the values of CM and CF increase since it increases the possibility of generating tasks of high utilization. We also can see that the results of the considered techniques show the similar trend to Fig. 5; UB-H&L provides an upper-bound, and the rest of the three techniques produce distinctive results.

Then, we show performance of each schedulability analysis technique (except UB-HL) over varying value of C_{ft}. Figure 8 plots the weighted schedulability of each technique according to an increasing value of C_{ft} from 0 to 20. As shown in Fig. 8, the weighted schedulability of AMC-p and AMC-non remain constant for a changing value of C_{ft} due to no consideration of a security constraint, which shows about 32% performance gap consistently. Such gap is mainly from that AMC-non considers a maximum blocking time of a lower priority job (i.e., B_i) unlike AMC-p does not. Note that such handicap is originated from a characteristic of non-preemptive scheduling itself, which is not avoidable to a schedulability analysis thereof. Despite such disadvantage, AMC-mf and AMC-ob outperform AMC-ob-p for large values of C_{ft}. For an increasing value of C_{ft} from 0

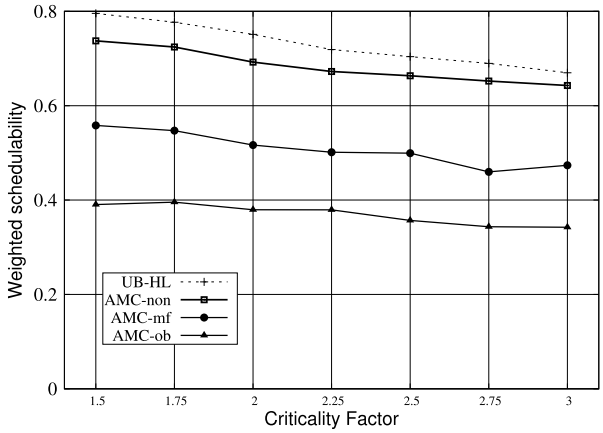


Fig. 6 Varying the criticality factor ($C_{ft} = 5, CM = 0.5$).

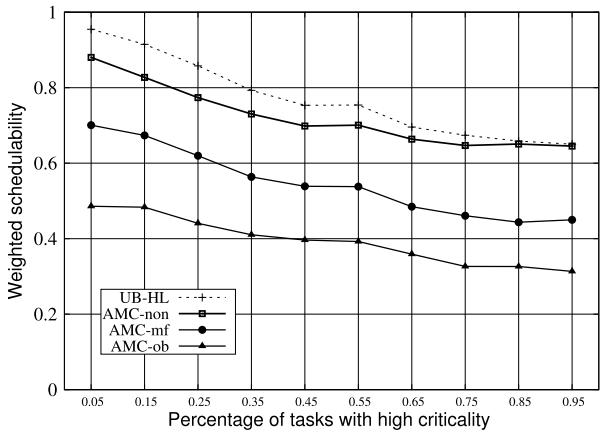


Fig. 7 Varying the number of HI-criticality tasks ($C_{ft} = 5, CF = 2$).

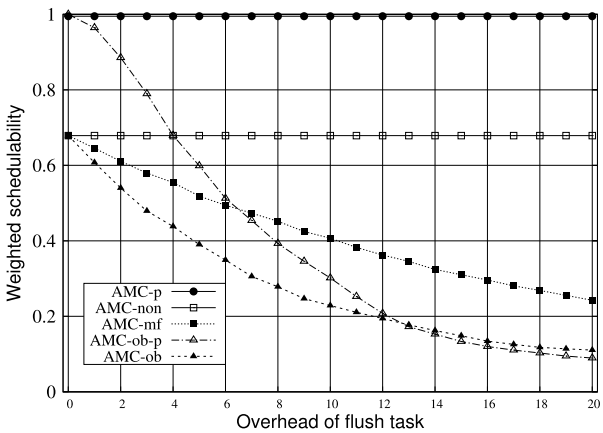


Fig. 8 Varying the overhead of flush task ($CF = 2, CM = 0.5$).

to 20, weighted schedulability of AMC-ob-p sharply drops from about 1.0 to 0.1. On the other hand, those of AMC-mf and AMC-ob decrease at relatively lower rates both from about 0.68 but to 2.5 and 0.12 respectively, which outperform AMC-ob-p for a value of C_{ft} larger than 7 and that of 13 respectively. This trend is shown because AMC-ob-p upper-bounds the number of FTs much naively $(2 \cdot N_{hep(i)} + 1)$ than AMC-mf and AMC-ob, enlarging the performance gap between AMC-ob-p, and AMC-mf and AMC-ob as a value of C_{ft} increases.

7.3 Discussion

We discuss how to improve performance of AMC-mf and AMC-ob-p independently. As we presented in previous section, our proposed schedulablity analysis (represented by AMC-mf) upper-bounds the maximum number of FTs after a mode change with I^a_{ft}, which is the most pessimistic factor of the analysis. That is, since I^a_{ft} considers interfering jobs in $hepH(i)$ in the interval $[0, R_i^{HI+})$ to construct a FT graph while I^b_{ft} does it in the interval $[0, R_i^{LO+})$ independently, some jobs in $hepH(i)$ in the intersected interval of $[0, R_i^{HI+})$ and $[0, R_i^{LO+})$ are considered twice. We will achieve a significant improvement of AMC-mf if we can exclude as many jobs in $hepH(i)$ as possible from the consideration of I^a_{ft} via much detailed reasoning regarding I^a_{ft}.

When it comes to preemptive FP scheduling for the AMC scheme, which incorporates a security constraint, we believe that a similar underlying principle of AMC-mf (bounding the maximum number of FTs occurring before and after a mode change separately) is applicable to derive a better performing schedulabilty analysis than AMC-ob-p. Such schedulablity analysis may need a new type of FT graph that can captures characteristics of preemptive scheduling to upper-bound the maximum number of FTs. A study proposed in [2] exploiting another FT graph well capture such characteristics gives a hint to extend our work into a class of preemptive scheduling. We leave it as a future work.

8. Related Work

A body of studies for which the real-time requirement and security mechanisms are combined already exists [5], [6]. Xie and Lin considered periodic task scheduling for which security service is required whereby varying overheads were applied in relation to the level of service. They improved the conventional scheduler to satisfy the real-time requirement and proposed a new scheduler to maximize the level of security service. The problem of information leakage in real-time database systems has also been studied [25], [26].

In early work, the leakage of information through shared resource in real-time systems was addressed [2], [3]. They suggested a flushing mechanism to clean up the state of the shared resource to prevent the information leakage and real-time scheduling techniques recognizing such security mechanisms. The initial research [3] proposed a restrictive security model and was aimed at a study of the initial tradeoffs between security requirements and real-time guarantees. The initial work was extended to a more general model, and a realistic application case study (for UAVs) was studied [2].

The number of FT invocations hindering a job under

analysis is closely related to the number of preemptions occurred during the execution of a job. A previous study proposed a mechanism exactly counting the number of preemptions for preemptive FP scheduling [27]. When a task is preempted by a higher priority task, a FT should be invoked if the higher priority task has lower security-level to prevent information leakage. Thus, the previous work gave many hints to calculate the exact number of FTs in a schedulability analysis.

In our previous study [4], we improved the existing mechanism proposed in [27] to be able to count the exact number of FTs. We suggested a new FP scheduling algorithm called Lowest Security-level First (LSF) and an intelligent FT invocation mechanism reserving FTs not in a greedy manner, which is called FT reservation. The combination of these two mechanisms derives an exact schedulability analysis. However, the proposed analysis is applied to LSF scheduling algorithm only, which shows limited analytic capability.

9. Conclusion

In this paper, we addressed the problem of the information leakage that can occur between real-time tasks having different levels of security in an MCS. We first defined the concept of a security constraint for an MCS employing the mechanism of FT and then proposed a new real-time scheduling algorithm and a new schedulability analysis for the AMC scheme incorporating the constraint. From the experimental results, our proposed mechanisms showed acceptable performance overheads while satisfying a security constraint to mitigate information leakage.

In the future, we will relax the restriction of non-preemptiveness of this work since our proposed work is limited to non-preemptive scheduling; we intend to extend this work to be applicable to both of preemptive and non-preemptive scheduling. We also plan to consider scheduling and analysis mechanisms for the multi-processor platforms.

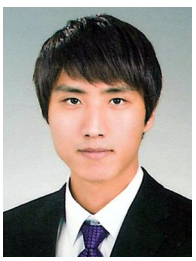
Acknowledgements

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-2014R1A1A1035827). This research was also supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.R0190-15-2071, Open PNP platform for diversity of autonomous vehicle based on cloud map). This research was also supported by the MISP (Ministry of Science, ICT & Future Planning), Korea, under the National Program for Excellence in SW (R2215-16-1005) supervised by the IITP (Institute for Information & communications Technology Promotion).

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," IEEE Symposium on Security and Privacy (SP), pp.447–462, 2010.
[2] R. Pellizzoni, N. Paryab, M.-K. Yoon, S. Bak, S. Mohan, and R.B. Bobba, "A generalized model for preventing information leakage in hard real-time systems," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp.271–282, 2015.
[3] S. Mohan, M.K. Yoon, R. Pellizzoni, and R. Bobba, "Real-time systems security through scheduler constraints," Euromicro Conference on Real-Time Systems (ECRTS), pp.129–140, 2014.
[4] H. Baek, J. Lee, Y. Lee, and H. Yoon, "Preemptive real-time scheduling incorporating security constraint for cyber physical systems," IEICE Trans. Inf. & Syst., vol.E99-D, no.8, pp.2121–2130, Aug. 2016.
[5] M. Lin, L. Xu, L.T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," IEEE Trans. Ind. Informat., vol.5, no.1, pp.22–37, 2009.
[6] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," ACM Trans. Embedd. Comput. Syst., vol.6, no.3, Article No. 20, 2007.
[7] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," IEEE International Real-Time Systems Symposium (RTSS), pp.239–243, 2007.
[8] S. Baruah, "Response-time analysis for mixed criticality systems," IEEE International Real-Time Systems Symposium (RTSS), pp.34–43, 2011.
[9] S. Baruah and A. Burns, "Implementing mixed criticality systems in Ada," Proc. Reliable Software Technologies - Ada-Europe, Lecture Notes in Computer Science, vol.6652, pp.174–188, Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
[10] M. Grenier and N. Navet, "Fine-tuning MAC-level protocols for optimized real-time QoS," IEEE Trans. Ind. Informat., vol.4, no.1, pp.6–15, 2008.
[11] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemption points," IEEE International Real-Time Systems Symposium (RTSS), pp.212–224, 2006.
[12] H. Ramaprasad and F. Mueller, "Tightening the bounds on feasible preemptions," ACM Trans. Embed. Comput. Syst., vol.10, no.2, pp.1–34, 2010.
[13] G. Buttazzo and A. Cervin, "Comparative assessment and evaluation of jitter control methods," International Conference on Real-Time and Network Systems (RTNS), pp.137–144, 2007.
[14] A. Burns and R.I. Davis, "Adaptive mixed criticality scheduling with deferred preemption," IEEE Real-Time Systems Symposium (RTSS), pp.21–30, 2014.
[15] M. Joseph and P. Pandya, "Finding response times in a real-time system," The Computer Journal, vol.29, no.5, pp.390–395, 1986.
[16] A. Burns, "The application of the original priority ceiling protocol to mixed criticality systems," IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp.7–11, 2013.
[17] Q. Zhao, Z. Gu, and H. Zeng, "Integration of resource synchronization and preemption-thresholds into EDF-based mixed-criticality scheduling algorithm," IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp.227–236, 2013.
[18] Q. Zhao, Z. Gu, and H. Zeng, "PT-AMC: Integrating preemption thresholds into mixed-criticality scheduling," Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.141–146, 2013.
[19] P.C. Kocher, "Timing attack 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, Berlin, Heidelberg, 1996.
[20] R. Mancuso, R. Dudko, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni, "Real-time cache management framework for multi-core architectures," IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp.45–54, 2013.
[21] Z.P. Wu, Y. Krish, and R. Pellizzoni, "Worst case analysis of DRAM latency in multi-requestor systems," IEEE Real-Time Systems Symposium (RTSS), pp.372–383, 2013.
[22] C.L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of ACM, vol.20, no.1, pp.46–61, 1973.
[23] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," Software Engineering Journal, vol.8, no.5, pp.284–292, 2002.
[24] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," International Workshop on Operating Systems Platforms for Embedded Real-Time Systems Symposium, pp.16–78, 2010.
[25] Q.N. Ahmed and S.V. Vrbsky, "Maintaining security in firm real-time database systems," Conference on Computer Security Applications, pp.83–90, 1998.
[26] C. Percival, "Cache missing for fun and profit," Proc. BSDCan, 2005.
[27] P.M. Yomsi and 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.



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.

Jinkyu Lee is an assistant professor in Department of Computer Science and Engineering, Sungkyunkwan University (SKKU), Republic of Korea, where he joined in 2014. He received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea, in 2004, 2006, and 2011, respectively. He has been a visiting scholar/research fellow in the Department of Electrical Engineering and Computer Science, University of Michigan, U.S.A. in 2011–2014. His research interests include system design and analysis with timing guarantees, QoS support, and resource management in real-time embedded systems, mobile systems, and cyber-physical systems. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011, and the Best Paper Award from the 33rd IEEE Real-Time Systems Symposium (RTSS) in 2012.