Improving Security for Periodic Tasks in Embedded Systems through Scheduling

TAO XIE
San Diego State University
and
XIAO QIN
New Mexico Institute of Mining and Technology

While many scheduling algorithms for periodic tasks ignore security requirements posed by sensitive applications and are, consequently, unable to perform properly in embedded systems with security constraints, in this paper, we present an approach to scheduling periodic tasks in embedded systems subject to security and timing constraints. We design a necessary and sufficient feasibility check for a set of periodic tasks with security requirements. With the feasibility test in place, we propose a scheduling algorithm, or SASES (security-aware scheduling for embedded systems), which accounts for both security and timing requirements. SASES judiciously distributes slack times among a variety of security services for a set of periodic tasks, thereby optimizing security for embedded systems without sacrificing schedulability. To demonstrate the effectiveness of SASES, we apply the proposed SASES to real-world embedded systems such as an automated flight control system. We show, through extensive simulations, that SASES is able to maximize security for embedded systems while guaranteeing timeliness. In particular, SASES significantly improves security over three baseline algorithms by up to 107%.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Real-time systems, security-sensitive applications, periodic tasks, scheduling, embedded systems

ACM Reference Format:

Xie, T. and Qin, X. 2007. Improving security for periodic tasks in embedded systems through scheduling. ACM Trans. Embedd. Comput. Syst. 6, 3, Article 20 (July 2007), 19 pages. DOI = 10.1145/1275986.1275992 http://doi.acm.org/ 10.1145/1275986.1275992

This research was supported in part by the New Mexico Institute of Mining and Technology under Grant 103295 and by Intel Corporation under Grant 2005-04-070.

Authors' addresses: Tao Xie, Department of Computer Science, San Diego State University, San Diego, CA 92182; email: xietao@nmt.edu; Xiao Qin, Department of Computer Science, New Mexico Institute of Mining and Technology, Socorro, NM 87801; email: xqin@cs.nmt.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1539-9087/2007/07-ART20 $5.00 DOI 10.1145/1275986.1275992 http://doi.acm.org/ 10.1145/1275986.1275992

1. INTRODUCTION

Embedded systems, ranging from intelligent vehicle highway system [Godbole et al. 1994] and hearing aids [Dijk et al. 1998] to satellite [Kawano and Kudoh 2001] and the electrical power grid [Salmeron et al. 2004], have been applied to diverse environments, including real-time computing platforms, which depend not only on results of computation, but also on time instants at which these results become available [Maheswaran and Siegel 1998; Zhang and Sivasubramaniam 2001]. The consequences of missing deadlines of hard real-time systems may be catastrophic, whereas such consequences for soft real-time systems are relatively less damaging. Examples of hard real-time applications include aircraft control [Faller and Schreck 1995], radar for tracking missiles [Mahafza et al. 1998], and medical electronics [Suzuki et al. 2002]. Online transaction processing systems are examples of soft real-time applications [Nilsson and Dahlgren 1999].

Since many embedded systems need to access, store, and manipulate security-sensitive data [Ravi et al. 2004], improving quality of security in embedded systems is increasingly becoming a critical and challenging issue in the design and development of embedded, real-time systems. Although there exists a large body of research related to security in the context of general-purpose computing systems, security techniques developed for PCs or servers are inadequate for embedded systems. Securing real-time embedded systems relies on a careful selection of security strategies, which are, in most cases, computational intensive and likely to push computing resources to the limit.

Today there are a variety of systems that have real-time and security considerations, because sensitive data and processing require special safeguard and protection against unauthorized access [Son et al. 2000]. In particular, real-time applications running in embedded systems require security protections to completely fulfill their trustworthy computing needs. However, conventional real-time systems, which are developed to guarantee timing constraints, while possibly posing unacceptable security risks, fail to meet the requirements of information security and assurance for modern real-time applications.

In this paper, we aim to develop fundamental and innovative real-time scheduling algorithms that are intended to achieve high quality of security for embedded systems while improving resource utilization. In particular, we propose a real-time scheduling algorithm, or SASES (security-aware scheduling for embedded systems). SASES seamlessly integrates security requirements into scheduling for real-time applications running in embedded systems.

The rest of the paper is organized as follows. Section 2 includes a summary of related work in this area. Section 3 presents the preliminary system architecture and task model. In Section 4, we propose a real-time scheduling algorithm for the periodic task model. To demonstrate the effectiveness of our scheduling algorithm, we present, in Section 5, the performance results based on a real-world embedded system under various workload conditions. Finally, Section 6 concludes the paper with summary and future research directions.

2. RELATED WORK

Scheduler is an important part of an embedded system and various scheduling techniques were reported in the literature. Grajcar [2000] made use of a dataflow model to schedule a computation, including conditional branches in an embedded system. Kadayif et al. [2002] proposed a locality-conscious scheduling algorithm to potentially reuse data among processes. However, these algorithms are unable to support real-time applications.

Real-time scheduling is a key factor in obtaining high performance and predictability for embedded systems. Real-time scheduling algorithms are intended to map tasks onto embedded processors and order tasks' execution in a way that deadlines are satisfied [Qin and Jiang 2001]. Various aspects of complicated real-time scheduling problems were addressed in the literature. In general, real-time scheduling algorithms fall into two camps: static [Abdelzaher and Shin 1999; Qin et al. 2000] and dynamic [Cheng and Huang 2004; Kalogeraki et al. 2000]. Conventional real-time scheduling algorithms like rate monotonic (RM) algorithm [Liu and Layland 1973], earliest deadline first (EDF) [Stankovic et al. 1998], and spring scheduling algorithm [Ramamritham and Stankovic 1984] were successfully applied in real-time systems. We proposed static Qin et al. [2000, 2003] and dynamic [Qin and Jiang 2001] scheduling and load-balancing schemes. Since many embedded, real-time systems provide security sensitive functions that are likely to be sabotaged by malicious entities, security is one of the most important issues to be addressed in embedded systems design. Most of existing scheduling algorithms perform poorly for security-sensitive real-time applications because of the ignorance of security requirements posed by the applications.

To protect embedded systems against all possible security threats, increasing attention has been made to security awareness in the context of embedded systems. Sen et al. [2003] proposed a cryptosystem based on cellular automata to ensure security of embedded systems. Ravi et al. [2004] analyzed challenges involved in secure embedded system design. Shao et al. [2004] proposed a hardware/software-defending technique to protect embedded systems against buffer overflow attacks. However, the above security techniques are not appropriate for real-time applications because of the lack of ability to express and handle timing constraints.

Some research has been conducted to factor in security concerns in a variety of embedded, real-time systems [Ahmed and Vrbsky 1998; George and Haritsa 1997; Rao 1989]. Rao [1989] developed security audit subsystems for real-time embedded avionics systems. George and Haritsa [1997] proposed concurrency control protocols, which can be used to support applications with real-time and security requirements. Ahmed and Vrbsky [1998] developed a secure optimistic concurrency control protocol that makes trade-offs between security and real-time requirements. Son et al. [2000] proposed a new scheme to improve timeliness by allowing partial violations of security. Muresan et al. [2005] introduced a novel system-on-chip architecture that prevents sensitive information from disclosing by controlling, in real-time, the power and current consumption of an embedded system. Saputra et al. [2005] proposed an architecture used to provide selective encryption protection for storage and processing protection to attacks for security-sensitive data. Our work is fundamentally different from the above approaches because ours is intended to develop a scheduling algorithm, which can meet security constraints in addition to real-time requirements of periodic tasks running in embedded systems.

In our previous work, we proposed a family of dynamic security-aware scheduling algorithms for clusters [Xie et al. 2005; Xie and Qin 2005b, 2006], and a grid [Xie and Qin 2005a]. These scheduling algorithms only support aperiodic tasks and, therefore, we are motivated in this study to propose a scheduling algorithm to improve security of a set of periodic tasks.

3. SYSTEM MODEL

In this study, we consider periodic real-time tasks, because a variety of real-time tasks are periodic in nature. For example, real-time tasks that receive, process, and transmit video, audio, and images are, in most cases, periodic tasks. It is assumed that periodic tasks are independent of one other [Thomadakis and Liu 1999]. A periodic real-time task is modeled as a set of parameters, e.g., $T_i = (e_i, p_i, l_i, S_i)$, where e_i is the worst-case execution, p_i is the period, and l_i denotes the amount of data (measured in KB) to be protected. S_i is a vector of security requirements. Note that e_i can be estimated by code profiling and statistical prediction. Without loss of generality, we assume that the first instance of T_i is ready for execution at time 0. Task T_i generates a new task instance every p_i time units, and the jth instance of T_i is invoked at time $(j-1)p_i$. The deadline of the jth task instance is equal to the ready time of the next instance, e.g., at time jp_i. It is supposed that T_i, requires q security services denoted by a vector of security-level ranges $S_i = (S_i^1, S_i^2, \ldots, S_i^q)$, where S_i^j is the security-level range of the jth security service. Values of security levels are normalized to the range from 0 to 1. Our scheduler is intended to determine the most appropriate point s_i in space S_i, e.g., $s_i = (s_i^1, \ldots, s_i^q)$, where $s_i^j \in S_i^j, \quad 1 \leq j \leq q$.

Since high security is achieved at the cost of performance degradation, we have to consider security overhead posed by security services. In the following security-overhead model, we consider confidentiality, integrity, and authentication, which are three security services widely deployed in embedded systems. The security-overhead model is general in the sense that the model can be easily extended to incorporate more security services.

Again, we assume that task T_i requires q security services provided in a sequential order. Let $c_{ij}^k(s_{ij}^k)$ be the overhead of the kth security service for the jth task instance, the security overhead c_{ij} experienced by the jth instance of T_i, can be computed using Eq. (1).

$$c_{ij} = \sum_{k=1}^{q} c_{ij}^k\left(s_{ij}^k\right), \text{ where } s_{ij}^k \in S_i^k. \tag{1}$$

Table I. Authentication Overhead

Authentication Methods	s_i^a :Security Level	$c_i^a(s_i^a)$:Computation Time (ms)
HMAC-MD5	0.25	90
HMAC-SHA-1	0.50	148
CBC-MAC-AES	0.75	163
Kerberos	1.0	3,060,000

The security overhead of the jth instance of T_i requesting for the aforementioned three services can be modeled by the following equation:

$$c_{ij} = \sum_{k \in \{a,\, e,\, g\}} c_{ij}^k\left(s_{ij}^k\right), s_{ij}^k \in S_i^k \tag{2}$$

where $c_{ij}^e(s_{ij}^e)$, $c_{ij}^g(s_{ij}^g)$, and $c_{ij}^a(s_{ij}^a)$ are overheads caused by the authentication, confidentiality, and integrity services [Xie et al. 2005]. The authentication overhead can be obtained from Table I [Elkeelany et al. 2002].

Based on the authentication methods' performance, each authentication approach is assigned a normalized security level in the range (0, 1]. Since, in our study, we consider four different mechanisms for authentication service, the strongest one is assigned security level 1 while the security level of the weakest mechanism is set to 0.25 (i.e., level 0.25 implies that we use HMAC-MD5, which is the weakest, yet fastest, method among the alternatives).

The confidentiality overhead c_{ij}^e is computed using Eq. (3), where π_i^e is the CPU time spent in encrypting security sensitive data.

$$c_{ij}^e\left(s_{ij}^e\right) = \pi_i^e s_{ij}^e, \text{ where } s_i^e \in S_i^e \tag{3}$$

The integrity overhead can be calculated using the following equation, where l_i is the amount of security sensitive data, and $\mu^g(s_i^g)$ is a function mapping a security level into its corresponding integrity service performance.

$$c_{ij}^g\left(s_{ij}^g\right) = l_i \Big/ \mu^g\left(s_{ij}^g\right) \tag{4}$$

4. THE SASES ALGORITHM

The main goal of the proposed scheduling algorithm is to maximize security of periodic tasks while meeting timeliness constraints. To achieve this goal, the SASES algorithm utilizes the following function to measure the security benefits gained by a task instance. Specifically, the security benefit of the jth instance of task T_i is modeled as a security-level function denoted by SL: $S_i \rightarrow \Re$, where $\Re$ is the set of positive real numbers:

$$SL(s_{ij}) = \sum_{k=1}^{q} w_i^k s_{ij}^k, \text{ where } 0 \le w_i^k \le 1 \quad \text{and} \quad \sum_{j=1}^{q} w_i^k = 1. \tag{5}$$

where $w_i^k$is the weight of the kth security service for task T_i. Users specify in their requests the weights to reflect relative priorities given to the required security services.

The scheduling decision of the jth instance of task T_i is feasible if (1) its deadline $d_{ij} = jp_i$ can be met and (2) all the security requirements are satisfied.

It has been proved that there exists a feasible schedule for a set of periodic tasks if, and only if, there is a feasible schedule for the planning cycle of the tasks [Hou and Shin 1997]. Note that the planning cycle p is the least common multiple of all the tasks' periods. Given a periodic real-time task T_i, there are p/p_i instances of T_i in a planning cycle p and the security benefit of each instance of T_i can be calculated using Eq. (5). Thus, by adding security benefits of p/p_i instances together, we can obtain the following nonlinear optimization problem formulation to compute the optimal security benefit of T_i in one planning cycle:

$$\begin{aligned} \text{maximize } SL_i &= \sum_{j=1}^{p/p_i} \sum_{k=1}^{q} w_i^k s_{ij}^k \\ \text{subject to } \min\left(S_i^k\right) &\le s_{ij}^k \le \max\left(S_i^k\right) \\ f_{ij} &< jp_i, \end{aligned} \tag{6}$$

where f_{ij} is the finish time ofthe jth instance of T_i, and min (S_i^j) and max (S_i^j) are the minimum and maximum security requirements of task T_i. Note that the above express includes both security and timeliness constraints.

The SASES algorithm attempts to maximize the system's security value defined as the sum of the security levels of all the tasks. The following security value function has to be optimized, subjecting to certain timing and security constraints:

$$\text{maximize } SV = \sum_{i=1}^{n} SL_i = \sum_{i=1}^{n} \sum_{j=1}^{p/p_i} \sum_{k=1}^{q} w_i^k s_{ij}^k$$

$$\text{subject to } \min\left(S_i^k\right) \le s_{ij}^k \le \max\left(S_i^k\right) \tag{7}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{p/p_i} \left[e_i + \sum_{k=1}^{q} c_i^k\left(s_{ij}^k\right) \right] \le p \tag{8}$$

$$f_{ij} < jp_i, \tag{9}$$

where n is the number of periodic tasks in the task set and p is the planning cycle. The first constraint encodes the fact that the security requirements have to be met. The second constraint indicates that the total computing demand is not allowed to exceed the available processor capacity. The last constraint states that the deadlines must be guaranteed.

To maximize security of all tasks, SASES selects the most appropriate security levels for tasks in a way that the tasks' deadlines and periods are unaffected. We now propose a way of checking feasibility, which is used to verify that whether it is possible to complete all tasks within timing constraints under selected security levels. Given a set of tasks with security requirements, the following proposition presents a necessary and sufficient feasibility check.

THEOREM 1. *Given a task set $T=\{T_1, \ldots, T_n\}$ and a set of security levels $s=\{s_1, \ldots, s_n\}$, all tasks can be feasibly scheduled on one processor if, and only if,* *the total utilization is equal to or less than one. That is,*

$$\sum_{i=1}^{n}\left[\left(e_i+\sum_{k-1}^{q}c_i^k\left(s_i^k\right)\right)\Big/ p_i\right]\le 1,\ \textit{where}\ \ s_i=\left(s_i^1,s_i^2,\ldots,s_i^q\right)$$

Proof. It has been proved in the necessary and sufficient feasibility check for a task set under ideal EDF scheduling requires the total of the worst-case utilizations be equal to or less than one, e.g., $\sum_{i=1}^{n}\frac{tc_i}{p_i}\le 1$ [Liu and Layland 1973], where tc_i is the total execution time of the task. tc_i equals to $e_i+\sum_{k=1}^{q}c_i^k(s_i^k)$ where$\sum_{k=1}^{q}c_i^k(s_i^k)$ is the security overhead of task t_i. After replacing tc_i by $e_i+\sum_{k=1}^{q}c_i^k(s_i^k)$, we obtain the necessary feasibility test for security sensitive tasks as $\sum_{i=1}^{n}[(e_i+\sum_{k}^{q}c_i^k(s_i^k))/p_i]\le 1$. □

Theorem 1 indicates that if there exists at least one feasible schedule, then the total processor utilization should not exceed the available computing capacity. Hence, given a task set $T$and a security level set s, there is no feasible schedule for T under the security level set if $\sum_{i=1}^{n}[(e_i+\sum_{k-1}^{q}c_i^k(s_i^k))/p_i]>1$. Based on the previous theorem, it is clear that Corollary 1 holds.

Next we show that we can find a feasible schedule when the security level is $s=\{(s_{1j}^k),\ldots,(s_{nj}^k)\}$, where $s_{i1}^k=\cdots=s_{ir}^k, r=p/p_i$.

Lemma 1. *Given a task set $T=\{T_1,\ \ldots,\ T_n\}$ and a set of security levels $s=\{(s_{1j}^k),\ldots,(s_{nj}^k)\}$, if there exist a feasible schedule with s that maximize the security value, then we also can find a feasible schedule with $s'=\{({s'}_{1j}^k),\ldots,({s'}_{nj}^k)\}({s'}_{i1}^k=\cdots={s'}_{ir}^k={s'}_i^k=(\sum_{j=1}^{r}s_{ij}^k)/r,\quad r=p/p_i,\quad 1\le i\le n,\quad 1\le k\le q)$ that can also maximize the security value.*

Proof. As a premise $\forall 1\le j\le r:\min(S_i^k)\le s_{ij}^k\le\max(S_i^k)$. Because${s'}_i^k$ is the mean of $s_{i1}^k,\ldots,s_{ir}^k$, $s'=\{({s'}_{1j}^k),\ldots,({s'}_{nj}^k)\}$ satisfies the security constraint (see Expression 7). Since for all $i\in[1,n]$ we have $\sum_{j=1}^{r}\sum_{k=1}^{q}c_i^k(s_{ij}^k)=\sum_{j=1}^{r}\sum_{k=1}^{q}c_i^k({s'}_{ij}^k)=r\sum_{k=1}^{q}{s'}_i^k$, it is clear that substituting $s'=\{({s'}_{1j}^k),\ldots,({s'}_{nj}^k)\}$ for $s=\{(s_{1j}^k),\ldots,(s_{nj}^k)\}$ will not affect the capacity constraint (see Expression 8). In addition, replacing substituting $s=\{(s_{1j}^k),\ldots,(s_{nj}^k)\}$ by $s'=\{({s'}_{1j}^k),\ldots,({s'}_{nj}^k)\}$ will not decrease the security value. Therefore, a feasible schedule with the security level set $s'=\{({s'}_{1j}^k),\ldots,({s'}_{nj}^k)\}$ can optimize the security value.

The following theorem proves that given a task set $T$and a security level set s, there exists an optimal feasible schedule where the security level of the kth security service of the ith task is a constant at every instance.

Theorem 2. *Given a task set $T=\{T_1,\ \ldots,\ T_n\}$ and a set of security levels $s=\{s_1,\ \ldots,\ s_n\}$, there exists a feasible schedule, where (1) the security value is optimized, and (2) the security level of the kth security service of T_i is constant at every instance, e.g., $s_{ij}^k=s_{ij'}^k=s_i^k$, where $j\ne j'$ and $\min(S_i^k)\le s_i^k\le\max(S_i^k)$.*

PROOF. First we prove that the schedule is feasible. The capacity constraint is satisfied, e.g.,$\sum_{i=1}^{n}\sum_{j=1}^{p/p_i}[e_i+\sum_{k=1}^{q}c_i^k(s_{ij}^k)]\le p$, and this is equivalent to $\sum_{i=1}^{n}[(e_i+\sum_{k-1}^{q}c_i^k(s_i^k))/p_i]\le 1$, where $s_i\ =(s_i^1,s_i^2,\ldots,s_i^q)$. It indicates that an ideal EDF scheduling can be used to generate a feasible schedule for the task set, where the security level of the kth security service of T_i is constant at every instance. Therefore, the timing constraint (see Expression 9) is satisfied. In addition, Lemma 1 shows that we can manage to find an optimal schedule where the security level of kth security service of T_i is constant at every instance. In conclusion, we prove that there is an optimal feasible schedule where the security level of the kth security service of T_i is a constant at every instance. □

Theorem 2 reveals that finding appropriate security levels of security services of each task can maximize security of a task set. Thus we have the following corollary.

COROLLARY 1. *Given a task set* $T=\{T_1,\ \ldots,\ T_n\}$ *and a set of security levels* $s=\{(s_{1j}^k),\ldots,(s_{nj}^k)\}$, *the objective function of the SASES scheduling algorithm is:*

$$\text{maximize } SV=\sum_{i=1}^{n}SL_i=\sum_{i=1}^{n}\sum_{k=1}^{q}w_i^k s_i^k \tag{10}$$

$$\text{subject to } \min\left(S_i^k\right)\le s_i^k\le\max\left(S_i^k\right) \tag{11}$$

$$\sum_{i=1}^{n}\left\{\frac{p}{p_i}\left[e_i+\sum_{k=1}^{q}c_i^k\left(s_i^k\right)\right]\right\}\le p \tag{12}$$

In light of Corollary 1, we are in a position to present a scheduling algorithm for periodic tasks with security constraints. To efficiently maximize security of a task set, SASES judiciously distributes the slack time among an array of security services for the tasks. Since it is imperative to give higher priorities to security services with high weights and low security overhead, we define a benefit–cost ratio function θ_i^k measuring the increase of security level by unit security overhead. That is,

$$\theta_i^k=w_i^k\Delta s_i^k\big/\left(c_i^k\left(s_i^k+\Delta s_i^k\right)-c_i^k\left(s_i^k\right)\right),\ \text{for the } k\text{th service of task } i \tag{13}$$

where the numerators represent the weighted increase in the security level, whereas the denominators indicate the corresponding increase in security overhead.

SASES intentionally selects the best candidate security service of a task with the highest benefit–cost ratio. The best candidate is chosen, based on the following expression, which suggests that raising the security level of the k'th service for task $T_{i'}$ can ultimately achieve maximized security.

$$\theta_{i'}^{k'}=\max_{1\le i\le n,1\le k\le q}\left\{\theta_i^k\right\} \tag{14}$$

An overview of the SASES algorithm is shown in Figure 1. SASES aims at maximizing quality of security under the timing constraints (see

```
1.       for $i$ = 1 to $n$ do
1.1        for $k$ = 1 to $q$ do
1.2          $s_i^k = \min\{S_i^k\};$
2.       while $\sum_{i=1}^{n}\left[\left(e_i + \sum_{k-1}^{q} c_i^k(s_i^k)\right)/p_i\right] \le 1$, where $s_i = (s_i^1, s_i^2, ..., s_i^q)$ do (see Theorem 1)
2.1        select the ith task and the kth security service subject to (see Expression 14)
$w_{i'}^{k'}\Delta s_{i'}^{k'}/\left(c_{i'}^{k'}(s_{i'}^{K'} + \Delta s_{i'}^{k'}) - c_{i'}^{k'}(s_{i'}^{k'})\right) = \max_{1 \le i \le n, 1 \le k \le q}\left\{w_i^k \Delta s_i^k/\left(c_i^k(s_i^k + \Delta s_i^k) - c_i^k(s_i^k)\right)\right\}$
2.2          if $s_{i'}^{k'} + \Delta s_{i'}^{k'} \le \max\{S_{i'}^{j'l}\}$ then (see Expression 11)
2.2.1          increase security level $s_{i'}^{j'} \leftarrow s_{i'}^{j'} + \Delta s_{i'}^{j'}$;
2.2.2          for $j$ = 1 to $p/p_{i'}$ do
2.2.2.1          update the schedule of the instances of $T_{i'}$, e.g., the start times of the instances;
2.3          else mark $s_{i'}^{j'}$ so that the kth security service of task $T_{i'}$ will no longer be considered;
2.4          update the worst-case utilization $\sum_{i=1}^{n}\left[\left(e_i + \sum_{k-1}^{q} c_i^k(s_i^k)\right)/p_i\right]$;
       end while
```

Fig. 1. The SASES scheduling algorithm for periodic tasks.

Expression 11). In an effort to meet both timeliness and security requirements, SASES first attempts to satisfy the timeliness constraints by setting all security levels to the minimal values of the security requirements. In doing so, SASES can maintain a high schedulability, which reflects a strong ability of guaranteeing deadline constraints.

If the available computing capacity is exceeded, Step 2 fully utilizes the slack time to accommodate high security levels provided that the real-time requirements are met. Step 2.1 gives the highest priority to a security service and a task with the largest benefit–cost ratio (see Expression 13), distributing the slack time on the most appropriate task. To optimize the security levels of the tasks, Steps 2.2 makes an effort to increase security levels of the selected security service for the task chosen in Step 2.1. The schedule of all the instances of the selected task are updated in accordance with the increased security level (see Step 2.2.2), because start times of the instances largely depend on how the slack time is distributed. Before the schedule feasibility is analyzed using Theorem 1, Step 2.4 calculates the system utilization. The time complexity of SASES is given as follows.

THEOREM 3. *The time complexity of SASES is* $O(h(nq + p/\min_{1 \le i \le n}(p_i)))$, *where h is the number of times Step* 2 *is repeated, q is the number of security services, and p is the planning cycle.*

PROOF. Step 1 takes time $O(nq)$ to initialize the security levels of all the tasks. The time complexity of selecting the most appropriate security service of a task is *O(nq)* (see step 2.1). Step 2.2.2 takes time $O(p/\min_{1 \le i \le n}(p_i))$ to update the schedule for the instances of the selected tasks. The time complexity

Table II. Time Complexities of the Four Algorithms

Algorithms	Time Complexity
minsEDF	$O(nq)$
maxsEDF	$O(nq)$
rndsEDF	$O(nq)$
SASES	$O(h(nq + p/\min_{1\le i\le n}(p_i)))$

of Step 2, is $O(nq + p/\min_{1\le i\le n}(p_i))$. Therefore, the time complexity of SASES is $O(nq + h(nq + p/\min_{1\le i\le n}(p_i))) = O(h(nq + p/\min_{1\le i\le n}(p_i)))$. □

5. EXPERIMENT RESULTS

In the previous section, we proposed the SASESalgorithm, which integrates security requirements into task scheduling for embedded systems. We implemented a simulator to evaluate the performance of SASES using synthetic workload and a real-world application.

We compare the proposed SASES with three baseline algorithms, which are variants of EDF. Throughout this section, the baseline algorithms are referred to as minsEDF, maxsEDF, and rndsEDF. Note that the baseline algorithms are intended to schedule periodic tasks with security constraints. However, these algorithms fail in optimizing security of periodic tasks. The baseline algorithms are summarized as follows.

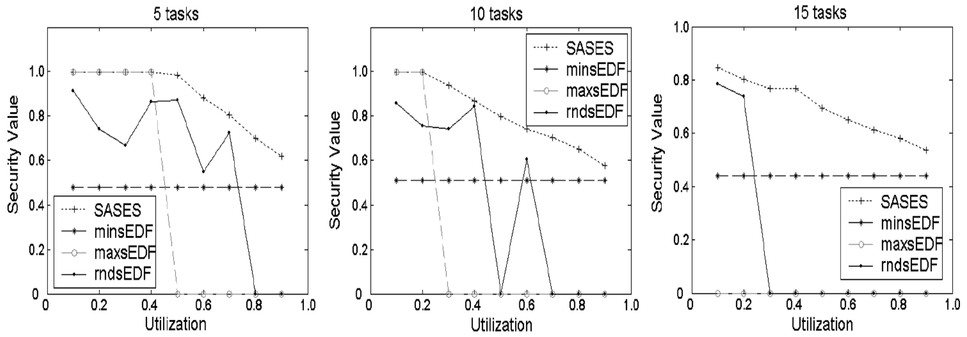
1. *minsEDF*: The scheduler selects the lowest security level of each security services required by a task. Thus, for the kth security service of T_i, the following equation is always held: $s_i^{v_l} = \min\{S_i^{v_l}\}$.
2. *maxsEDF*: The scheduler intentionally chooses the highest security level for each security requirement posed by each task. This fact can be encoded by the following expression: $\forall T_i, 1 \le k \le q : s_i^k = \max\{S_i^k\}$.
3. *rndsEDF*: rndsEDF randomly picks a value within the security level range for each requested security service of a task. Formally, the following expression is held in rndsEDF: $\forall T_i, 1 \le k \le q : s_i^k = \text{random}\{S_i^k\}$.

The time complexities of the three baseline algorithms are $O(nq)$ (see Theorem 3) because each of them only needs to execute an initialization step like Step 1 and ignores all the rest of the steps (see Figure 1). Table II lists the time complexities of all four algorithms evaluated in this section. Since h and $p/\min_{1\le i\le n}(p_i)$ (see Theorem 3), are all bounded values decided by a particular embedded system and the SASES algorithm is required to be executed only once to generate a feasible schedule; the time overhead of SASES is acceptable under most situations.

This section is organized as follows. Section 5.1 briefly describes the simulator. Section 5.2 is to examine the performance improvements of SASES over the three baseline algorithms. In Section 5.3, we will study the performance sensitivity of SASES to CPU capacities. We evaluate the impact of security-sensitive data size in Section 5.4. Section 5.5 presents the performance impact of security

Table III. Characteristics of System Parameters

Parameter	Value (fixed) - (varied)
CPU Speed	(100 million instructions/second or MIPS) – (100, 150, 200)
Task execution time	(100 ~ 1000) ms
Periods	(10 ~ 1000) ms
Mean size of data to be secured	(60, 90, 120) KB
Required security services	Confidentiality, integrity, and authentication
Weight of security services	(authentication : confidentiality : integrity) – (0.5, 0.3,0.2);(0.2,0.5,0.3); (0.2, 0.3, 0.5)



(a) (b) (c)

Fig. 2. Performance impact of number of tasks.

service weights. Finally, we validate results from the synthetic workload using a real application.

5.1 Simulator and Simulation Parameters

Table III summarizes the important configuration parameters of a simulated embedded system used in our experiments.

Note that the number of tasks, size of data to be secured, CPU capacities, and size of data to be secured are synthetically generated. However, we are able to study impacts of these workload parameters on system performance by controlling them as fundamental simulation parameters. Importantly, we ran a real application to validate results from the synthetic periodic tasks. The performance metric by which we evaluate system performance is *Security Value* (see Eq. 10).

5.2 Overall Performance Comparisons

The goal of this experiment is to compare the proposed SASES algorithm against the other three baseline schemes. Further, we tested the sensitivity of SASES to the number of tasks. We evaluated three simulated embedded systems with 5, 10, and 15 periodic tasks, respectively. Without loss of generality, we assume that each task requires three security services; and the mean size of the data to be secured is 60 KB.

Figure 2 shows results for these four algorithms on an embedded system where the CPU power is fixed at 100 MIPS. The period of each task is randomly generated in the range between 100 and 1000 ms. Similarly, the worst-case computation time of each task is uniformly distributed between 10 and 1000 ms. In addition, the computation times are adjusted in a way that the system utilization is set to a desired value. It is assumed that each instance of a task spends the worst-case computation time during its execution.

We observe from Figure 2a that SASES ties with maxsEDF in terms of security value when the utilization is smaller than 0.5, which implies that there is plenty of slack time that can be used to improve security levels of the tasks. Importantly, SASES significantly outperforms minsEDF during the utilization range between 0.1 and 0.5. It is observed that the performance of rndsEDF is not stable, because it always randomly selects security levels for each task, meaning that its performance is unpredictable. When the utilization exceeds 0.5, maxsEDF immediately fails in generating feasible schedules; thus, its security value degraded to zero. Similarly, rndsEDF also failed to schedule the task set with five tasks when utilization is over 0.8. This is mainly because there is not enough slack time to be exploited to improve tasks' security levels when the system utilization is increased.

Figure 2b plots security values of the four algorithms under the workload where there are ten tasks running in the embedded system. It reveals that SASES consistently performs the four alternatives. For example, compared with SASES, achieved improvement in security values over minsEDF and rndsEDF by averages of 57.9 and 91.3%, respectively. This is because minsEDF and rndsEDF are unable to employ slack times to promote security levels of tasks. We noticed that when the utilization was larger than 0.2, maxsEDF is incapable of scheduling the task set, whereas this situation only happened when the utilization is larger than 0.4 (Figure 2a). This phenomenon can be explained by the fact that more tasks compete with each other for the uniprocessor and, therefore, some of tasks may miss their deadlines when maxsEDF is used to generate schedules. Such performance deterioration becomes more marked when the system utilization is high (see Figure 2c).

Figure 2c demonstratively shows that (1) maxsEDF completely failed to generate any feasible schedule even when utilization is set as low as 0.1; (2) rndsEDF can only be applied to embedded systems when utilization is very low (e.g., lower than 0.3); (3) minsEDF maintains the same performance level; and (4) SASES delivers an excellent performance in security under workload conditions where the system is overloaded. More importantly, when system utilization increases, SASES gracefully degrades its performance, while delivering better performance compared with minsEDF under a very heavy workload. This is because SASES can judiciously distribute slack time among the set of tasks to improve the quality of security. When the slack time becomes tight, SASES lowers security levels to ensure that timing constraints can be met.

In short, SASES consistently outperforms the three alternatives in all the three cases. SASES demonstrates an ability to deliver high performance, even when embedded systems are overloaded. Besides, its performance is predictable, which is desired feature for embedded systems. These valuable characteristics show us that SASES can be successfully applied to real embedded systems (see Section 5.6).

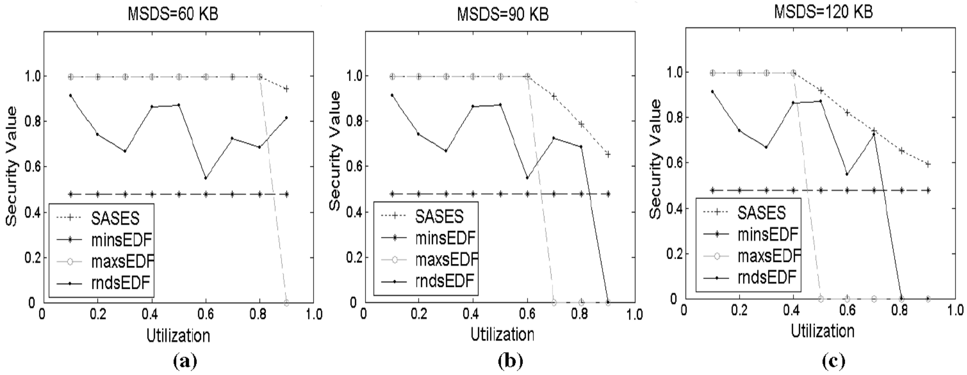


Fig. 3. Performance impact of SDS (size of data to be secured).

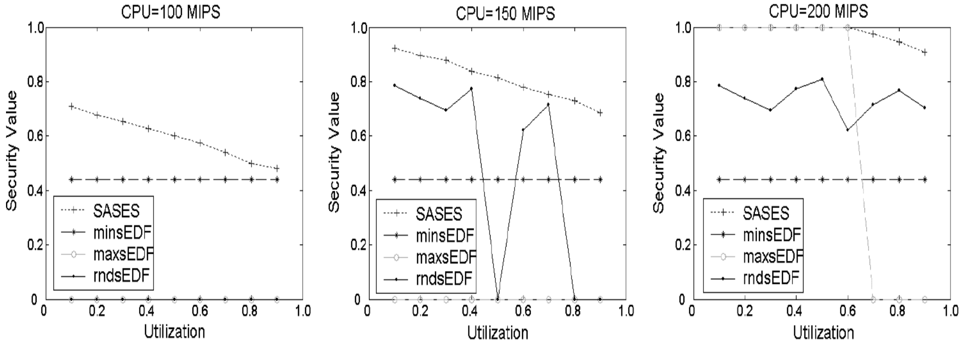
5.3 Impact of Size of Data to Be Secured

In this experiment, we evaluate the performance impact of size of data to be secured. The number of tasks is fixed to five and the CPU capacity is set to 100 MIPS. As we described in Section 5.1, each task was synthetically assigned a block of security-sensitive data that needs to be protected from being disclosed or tampered with. The data size influences task completion times, because the larger the data size, the longer time is needed to protect it. To discover the performance impact of the data size, we tested three cases where mean data size is 60, 90, and 120 KB, respectively.

The experimental results are shown in Figure 3. Several important observations are drawn from Figure 3. First, when the data size is varied from 60 to 90 KB, the security values of SASES, rndsEDF, and maxsEDF are decreasing, whereas that of minsEDF remains the same. This can be explained by the fact that for SASES, rndsEDF, and maxsEDF, increasing security overhead results in less slack time spend in improving security values. In many cases, rndsEDF and maxEDF are unable to yield feasible schedules because of the large security-sensitive data volume. For example, maxsEDF failed in scheduling the task sets when the utilization exceeds 0.9 (Figure 3a), 0.7(Figure 3b), and 0.5(Figure 3c), respectively. Interestingly, minsEDF can keep a constant performance in all three cases, because it always uses the minimal security level for each task. Note that minsEDF can produce feasible schedules in all the three situations, but minsEDF exhibits very low performance in terms of security.

Second, Figure 3 shows that SASES is sensitive to change of the data size as the security value decreases from 0.994 to 0.86. Nevertheless, SASES continuously outperforms the baseline algorithms in these three cases.

Third, it is interesting to observe that the performance improvements of SASES over rndsEDF and maxsEDF are increased with the increasing values of the data size. For example, SASES improves security values over maxsEDF by 11.8, 39.2, and 93.4%. Compared with rndsEDF, the improvements are 30.7, 38.6, and 44.9%. The improvements can be attributed to the rapid performance deterioration of rndsEDF and maxsEDF because of their incapability



(a) (b) (c)

Fig. 4. Performance impact of CPU capacity.

of generating feasible schedules when the overhead of securing data becomes significant. The performance degradation of SASES is more graceful than those of the alternatives. Compared with minsEDF, the performance improvement of SASES reduces from 107 to 79.1%, because minsEDF maintains an unchanged security level regardless of the value of data size.

5.4 Sensitivities to CPU Capacity

To examine performance sensitivities of the four algorithms to CPU capacity, we varied the CPU capacity from 100 to 200 MIPS with increments of 50 MIPS. In this set of experiments, the mean size of data to be secured is 90 KB and the number of tasks is 15. The results reported in Figure 4 reveals that SASES delivers better performance than the other alternatives. However, the performance discrepancy between SASES and maxsEDF is narrowed with the increasing CPU capacity. Figure 4 shows that SASES gradually degrades to maxsEDF when the computing capacity of the system is increased. This is mainly because when the system has adequate computing capacity, both SASES and maxsEDF are enabled to select the highest security levels for periodic tasks.

Figure 4a shows that neither rndsEDF nor maxsEDF can generate feasible schedules when there are 15 tasks sharing one 100 MIPS CPU. Conversely, SASES can provide feasible schedules for the task set, offering high security performance. Figure 4b indicates that SASES exhibits higher performance in the case where the CPU capacity increases by 50%. Figure 4c demonstrates that SASES potentially degrades to maxsEDF when the CPU capacity continuously increases. The conclusion is that the CPU capacity can greatly boost security performance, especially when the system is overloaded.

5.5 Impact of Security Service Weights

Recall that the security level model proposed in Section 4 is comprised of multiple security levels for various security services like authentication, confidentiality, and integrity. Each required service of a periodic task is assigned a weight, which reflects the priority of the service. To study the impact of security service weights on performance of SASES, we tested three configurations of security

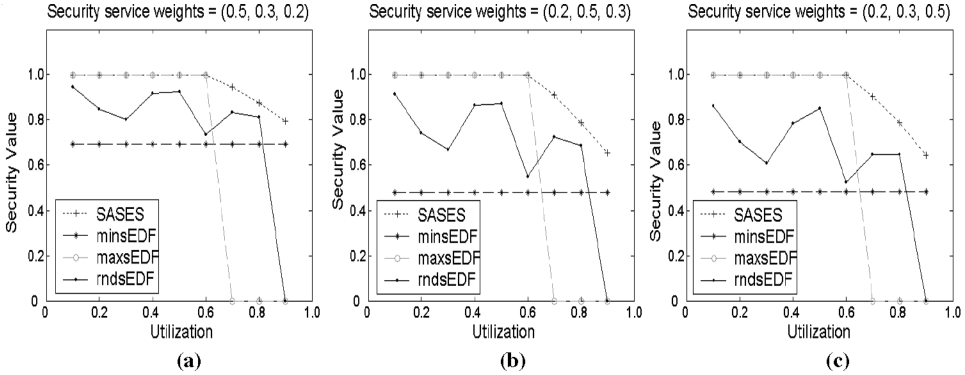


Fig. 5. Performance impact of security service weights.

service weights. In this set of experiments the data size was set to 90; the CPU capacity is 100 MIPS.

Results presented in Figure 5 show that security service weights impose little performance impact on SASES and maxsEDF. This is partially because the system load is not heavy and they can achieve the maximal or near optimal security performance in many cases. The second observation drawn from Figure 5 is that the performance of minsEDF is noticeably improved when the authentication weight is high. This result can be explained as follows. The security overhead caused by authentication is relatively lower compared with the other two security services. Hence, minsEDF can obtain the maximal security levels from the authentication service at the cost of little CPU time. The third observation is that Figure 5b is very similar to Figure 5c, meaning that the performance impact of the confidentiality service is almost identical to that of the integrity service. This result is reasonable because the security overhead caused by improving levels of the confidentiality service is close to the security overhead in increasing integrity levels.

5.6 Evaluation in Real Application

To validate the results from the synthetic simulations, we evaluated the SASES algorithm based on a real system—an automated flight control system [Abdelzaher et al. 2000]. The motivation of this experiment is to verify that SASES can outperform the baseline algorithms in the context of existing embedded systems.

The simulated aircraft flies around a pattern in which it executes a takeoff and climb, keeps a constant altitude around a rectangular course, and then descends through final approach and landing [Abdelzaher et al. 2000]. The mission goals are to complete the flight around the pattern and to destroy any observed enemy targets using onboard radar and missiles. Detailed information regarding the system, except for the size of data to be secured, can be found in Abdelzaher et al. [2000]. Table IV shows the set of parameters present for periodic tasks, including execution time, period, and three configurations of the data size.

Table IV. Task Model Parameters for Automated Flight Control System

Task	Exec. Time (ms)	Period (s)	Config1 (KB)	Config2 (KB)	Config3 (KB)
Guidance	100	1	300	400	500
Controller	80	5	300	400	500
Slow navigation	100	1	300	400	500
Fast navigation	60	0.2	300	400	500
Missile control	500	10	300	400	500

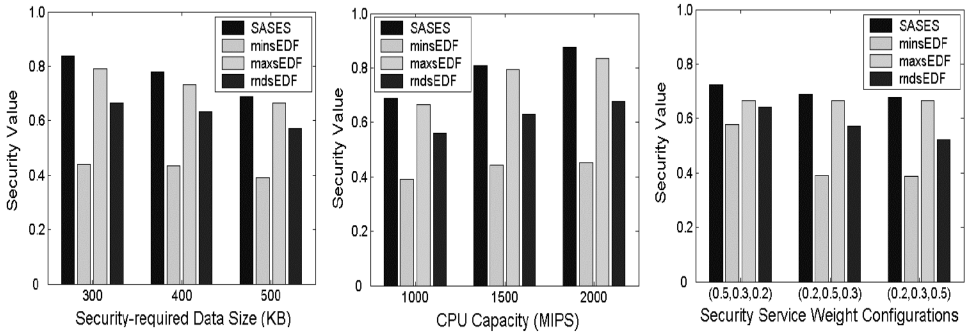
Table V. Experimental Parameters for Automated Flight Control System

Parameter	Value
CPU speed	(1000, 1500, 2000)MIPS
Required security services	Confidentiality, integrity, and authentication
Weight of authentication	0.2
Weight of confidentiality	0.5
Weight of integrity	0.3
Number of F-16 aircraft	1
Utilization	0.71

The automated flight control system was utilized to fly a simulated model of an F-16 fighter aircraft. In this system all the flight control tasks—including guidance, slow navigation, fast navigation, controller, and missile control—need to be executed in real time to meet their deadlines. The functions of these five periodic tasks are summarized as follows. The "guidance" task sets the reference trajectory of an aircraft in terms of altitude and heading; the "controller" is responsible for executing the closed-loop control functions that deal with actuator commands; the two "navigation" tasks read sensor values distinguished by the required update frequency; and finally, the "missile control" task is responsible for reading radar and firing missiles. These separate tasks are mandatory to control the aircraft during flight. It is assumed that (1) all tasks have known bounded execution times, (2) task arrivals are independent, (3) each task's deadline is equal to its period, and (4) tasks are nonpreemptive in nature [Abdelzaher et al. 2000].

Since the automated flight control system is applied in military battlefields, where security requirements, such as data confidentiality, are mandatory, we synthetically create different sizes of security-sensitive data during the execution of each version of the five tasks. The security requirements of each task include confidentiality, integrity, and authentication services. Each task instance experiences security overhead determined by Eq. (1). To evaluate the performance of SASES under various scenarios, we constructed three configurations of the data size (see columns 4–6 in Table IV). In Config1, we chose relatively low security levels for each task. Config2 represents medium security levels, whereas Config3 allocates relatively high security levels to the tasks. The experimental parameters for the automated light control system are summarized in Table V.

We conducted three experiments. In particular, the first one (Figure 6a) is focused on the size of security-sensitive data, the second one (Figure 6b) is intended to test the CPU capacity sensitivity of SASES in the real system, and the third one (Figure 6c) is to verify impact of the security service weights.



(a) (b) (c)

Fig. 6. Real application experiment.

Results plotted in Figure 6 are consistent with those reported in Figures 3–5, thereby verifying that SASES can gain performance improvements in the real world system.

Figure 6a shows that the mean security value of SASES decreases when the data size increases. This observation is consistent with the one drawn from Figure 3. Similarly, Figures 6b and 6c validate the results presented in Sections 6.4–6.5. It is worth noting that SASES marginally outperforms maxsEDF in this set of experiments. This is because (1) the system load is light (e.g., there are only five tasks), and (2) the CPU capacity is as high as 1000 MIPS. The high performance CPU implies that the security overhead is a nondominant part in the overall execution time of each task. As such, maxsEDF can achieve similar performance as SASES.

We consider the results discussed in this subsection to be a strong validation of our previous simulations, which suggest that SASES can be successfully applied to real embedded systems. In summary, the strength of SASES can be fully exhibited in a high workload situation, where 15 real-time tasks are running on a uniprocessor embedded system (see Section 5.2). When the workload is light, SASES will be gracefully degraded to maxsEDF. This fact hints that SASES can significantly improve security of real-world embedded systems without extra hardware cost; this is especially true when the embedded systems are overloaded.

6. CONCLUSIONS

This paper addressed the scheduling problem for periodic tasks with security and timing constraints. We designed a necessary and sufficient feasibility check, which can be used to test schedulability for a set of periodic tasks running in embedded systems. The feasibility check suggests that any periodic task scheduler that can achieve 100% CPU utilization is capable of generating optimal schedules. We proposed a scheduling algorithm, or SASES, which considers both security and timing requirements of periodic tasks. SASES distributes slack times among an array of security services for a set of periodic tasks. Therefore, SASES can optimize security for embedded systems without sacrificing schedulability. We conducted extensive simulation experiments to show that our SASES algorithm can consistently improve both security and performance over three alternatives. Specifically, our measurements showed that SASES achieves up to 107% improvement in security.

In our future work, we will focus on networked embedded systems, where communication overhead and network security must be factored in. Improving security for networked embedded systems is of critical importance, because a vast majority of networked embedded systems are inevitably exposed to the increasing threats from external and internal hackers. We will also develop dynamic scheduling algorithms for aperiodic task with various security requirements. A third future research direction is to extend the security overhead model to incorporate additional security services.

REFERENCES

Abdelzaher, T. F. and Shin., K. G. 1999. Combined task and message scheduling in distributed real-time systems. *IEEE Trans. Parallel and Distributed Sys.,* vol.10, no.11. (Nov.), 1179–1191.

Abdelzaher, T. F., Atkins, E. M., and Shin., K. G. 2000. QoS negotiation in real-time systems and its application to automated flight control, *IEEE Trans. Computers,* vol. 49, no. 11, (Nov.), 1170–1183.

Ahmed, Q. and Vrbsky, S. 1998. Maintaining security in firm real-time database systems. In *Proc. 14*th *Ann. Computer Security Application Conf.*

Cheng, S. and Huang, Y. 2004. Dynamic real-time scheduling for multi-processor tasks using genetic algorithm. In *Proc. 28th Ann. Int'l Conf. Computer Software and Applications*. 154–160.

Elkeelany, O., Matalgah, M., Sheikh, K., Thaker, M., Chaudhry, G., Medhi, D., and Qaddouri, J. 2002. Performance analysis of IPSec protocol: encryption and authentication. In *Proc. IEEE Int'l Conf. Communications*, New York, (April). 1164–1168.

Faller, W. E. and Schreck, S. J. 1995. Real-time prediction of unsteady aerodynamics: application for aircraft control and maneuverability enhancement. *IEEE Trans. Neural Networks 6*, 6, 1461–1468.

George, B. and Haritsa, J. 1997. Secure transaction processing in firm real-time database systems. In *Proc. ACM SIGMOD Conf.* (May).

Godbole, D. N., Lygeros, J., and Sastry, S. S. 1994. Hierarchical hybrid control: an IVHS case study. In *Proc. Conf. Control and Decision*. 1592–1597.

Grajcar, M. 2000. Conditional scheduling for embedded systems using genetic list scheduling. In *Proc. Int'l Symp. System Synthesis* (Sept.). 123–128.

Hou, C.-J. and Shin, K. G. 1997. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Trans. Computers 46*, 12, 1338–1356.

Kadayif, I., Kandemir, M., Kolcu, I., and Chen, G. 2002. Locality-conscious process scheduling in embedded systems. In *Proc. Int'l Symp. Hardware/Software Codesign*. 193–198.

Kalogeraki, V., Melliar-Smith, P. M., and Moser, L. E. 2000. Dynamic scheduling for soft real-time distributed object systems. In *Proc. 3rd IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*. 114–121.

Kawano, K. and Kudoh, J. 2001. Real-time satellite data transfer system for siberian NOAA image database. In *Proc. Int'l Symp. Geoscience and remote Sensing*, vol. 5 (July). 2277–2279.

Liu, C. L. and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM 20*, 1, 46–61.

Mahafza, B., Welstead, S., Champagne, D., Manadhar, R., Worthington, T., and Campbell, S. 1998. Real-time radar signal simulation for the ground based radar for national missile defense. In *Proc. the 1998 IEEE Radar Conf.* (May). 62–67.

Maheswaran, M. and Siegel, H. J. 1998. A Dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Proc. the 7th Heterogeneous Computing Workshop*. 57–69.

Muresan, R., Vahedi, H., Zhanrong, Y., and Gregori, S. 2005. Security-oriented application specific architectures: power-smart system-on-chip architecture for embedded cryptosystems. In

Proc. 3rd IEEE/ACM/IFIP Int'l Conf. Hardware/Software Codesign and System Synthesis. 184–189.

NILSSON, J. AND DAHLGREN, F. 1999. Improving performance of load-store sequences for transaction processing workloads on multiprocessors. In *Proc. Int'l Conf. Parallel Processing* (Sept.). 246–255.

QIN, X. AND JIANG, H. 2001. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems, In *Proc. 30th Int'l Conf. Parallel Processing,* Valencia, Spain, (Sept.). 113–122.

QIN, X., JIANG, H., XIE, C., AND HAN, Z. 2000. Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems. In *Proc. Int'l Conf. Parallel and Distributed Computing and Sys.* (Nov.).

QIN, X., JIANG, H., AND SWANSON, D. R. 2002. An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogeneous systems. In *Proc. Int'l Conf. on Parallel Processing*, (Aug.), British Columbia, Canada. 360–368.

QIN, X., JIANG, H., ZHU, Y., AND SWANSON, D. R. 2003. Dynamic load balancing for I/O-intensive tasks on heterogeneous clusters. In *Proc. the 10th Int'l Conf. High Performance Computing*, (Dec.). Hyderabad, India, 300–309.

RAMAMRITHAM, K. AND STANKOVIC, J. A. 1984. Dynamic task scheduling in distributed hard real-time system. *IEEE Software, 1*, 3 (July).

RAO, K. N. 1989. Security audit for embedded avionics systems. In *Proc. Annual Computer Security Applications Conference*. 78–84.

RAVI, S., RAGHUNATHAN, A., KOCHER, P., AND HATTANGADY, S. 2004. Security in embedded systems: design challenges. *ACM Trans. Embedded Computing Systems, 3*, 3, 461–491.

SALMERON, J., WOOD, K., AND BALDICK, R. 2004. Analysis of electric grid security under terrorist threat. *IEEE Trans. Power Systems, 19*, 2, 905–912.

SAPUTRA, H., OZTURK, O., VIJAYKRISHNAN, N., KANDEMIR, M., AND BROOKS, R. 2005. A data-driven approach for embedded security. In *Proc. IEEE Annual Symp. VLSI*. 104–109.

SEN, S., HOSSAIN, S. I., ISLAM, K., CHOWDHURI, D. R., AND CHAUDHURI, P. P. 2003. Cryptosystem designed for embedded system security. In *Proc. Int'l Conf. VLSI Design*. 271–276.

SHAO, Z., XUE, C., ZHUGE, Q., SHA, E. H.-M., AND XIAO, B. 2004. Security protection and checking in embedded system integration against buffer overflow attacks. In *Proc. Int'l Conf. Information Technology: Coding and Computing*, vol. 1. 409–413.

SON, S. H., MUKKAMALA, R., AND DAVID, R. 2000. Integrating security and real-time requirements using covert channel capacity. *IEEE Trans. Knowledge and Data Engineering*, *12*, 6, 865–879.

STANKOVIC, J. A., SPURI, M., RAMAMRITHAM, K., AND BUTTAZZO, G. C. 1998. Deadline scheduling for real-time systems—EDF and related algorithms. Kluwer Academic Publishers, Boston, MA.

SUZUKI, S., KATANE, T., SAOTOME, H., SAITO, O. 2002. Electric power-generating system using magnetic coupling for deeply implanted medical electronic devices. *IEEE Transactions on Magnetics, 38*, 5, 3006–3008.

THOMADAKIS, M. E. AND LIU, J.-C. 1999. On the efficient scheduling of non-periodic tasks in hard real-time systems. In *Proc. 20th IEEE Real-Time Systems Symp.* 148–151.

VAN DIJK, L. P. L. VAN DER, WOERD, A. C. MULDER, J., ROERMUND, AND A. H. M. VAN. 1998. An ultra-low-power, low-voltage electronic audio delay line for use in hearing aids. *IEEE Journal of Solid-State Circuits, 33*, 2, 291–294.

XIE, T. AND QIN, X. 2005a. Enhancing security of real-time applications on Grids through dynamic scheduling. In *Proc. 11th Workshop Job Scheduling Strategies for Parallel Processing*, (June), MA.

XIE, T. AND QIN, X. 2005b. A new allocation scheme for parallel applications with deadline and security constraints on clusters. In *Proc. 7th IEEE Int'l Conf. on Cluster Computing* (Sept.), Boston, MA.

XIE, T. AND QIN, X. 2006. SHARP: A new real-time scheduling algorithm to improve security of parallel applications on heterogeneous clusters. In *Proc. 25th IEEE Int'l Performance Computing and Communications Conf.* (Apr.), Phoenix, Arizona.

XIE, T., QIN, X., AND SUNG, A. 2005. SAREC: A security-aware scheduling strategy for real-time applications on clusters. In *Proc. the 34th Int'l Conf. Parallel Processing*, (June), Norway.

ZHANG, Y. AND SIVASUBRAMANIAM, A. 2001. Scheduling best-effort and real-time pipeline application on time-shared clusters. In *Proc. Int'l Symp Parallel Architecture and Algorithm*.

Received May 2005; revised January 2006; accepted March 2006