An Approach to Satisfying Security Needs of Periodic Tasks in High Performance Embedded Systems

Tao Xie Xiao Qin* Andrew Sung

Department of Computer Science
New Mexico Institute of Mining and Technology
Socorro, New Mexico 87801
{xietao, xqin, sung}@cs.nmt.edu

Abstract

Existing scheduling algorithms for periodic tasks ignore security requirements posed by sensitive applications and are consequently unable to perform properly in high performance embedded systems with security constraints. In this paper we present an approach to scheduling periodic tasks in high performance embedded systems subject to security and timing constraints. 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 high-performance embedded systems without sacrificing schedulability. We show through extensive simulations that SASES is able to maximize security for high-performance embedded systems while guaranteeing timeliness. In particular, SASES significantly improves security over three baseline algorithms by up to 93.4%.

1. Introduction

Since many high performance embedded systems need to access, store, and manipulate security sensitive data [8], 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 high performance 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 [9]. In particular, real-time applications running in high performance 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 networked 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.

2. System Model

In this study we consider periodic real-time tasks, because a majority 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 [10]. A periodic real-time task is modelled 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

* Contact author. *http://www.cs.nmt.edu/~xqin*

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 *j*th 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 $s_i^j \in S_i^j,\ 1 \le j \le q$.

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 encryption, 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 *k*th security service for the *j*th task instance, the security overhead c_{ij} experienced by the *j*th instance of T_i, can be computed using Equation (1).

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

The security overhead of the *j*th instance of T_i requesting for the aforementioned three services can be modelled by the following equation:

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

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, encryption, and integrity services [12]. The authentication overhead can be obtained using the following table.

Table 1. 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

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

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

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.

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

3. 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 *j*th 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:

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

w_i^k is the weight of the *k*th 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 *j*th 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 [3]. Note that the

planning cycle p is the least common multiple of all the tasks' periods. Given a periodic real-time task T_i, we can obtain the following non-linear optimization problem formulation to compute the optimal security benefit of T_i:

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

$$\text{subject to } \min(S_i^k) \le s_{ij}^k \le \max(S_i^k),$$

$$f_{ij} < jp_i, \tag{6}$$

where f_{ij} is the finish time of the 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:

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

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

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

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

where n is the number of periodic tasks in the task set, 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.

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** $\sum_{i=1}^{n} \left[\left(e_i + \sum_{k-1}^{q} c_i^k (s_i^k) \right) \Big/ p_i \right] \le 1$, where $s_i = (s_i^1, s_i^2, ..., s_i^q)$ **do**
2.1 select the i'th task and the k'th security service subject to
$w_{i'}^{k'} \Delta s_{i'}^{k'} / \left(c_{i'}^{k'} (s_{i'}^{K'} + \Delta s_{i'}^{k'}) - c_{i'}^{k'} (s_{i'}^{k'}) \right) = \max_{1 \le i \le n, 1 \le k \le q} \left\{ w_i^k \Delta s_i^k / \left(c_i^k (s_i^k + \Delta s_i^k) - c_i^k (s_i^k) \right) \right\}$;
2.2 **if** $s_{i'}^{k'} + \Delta s_{i'}^{k'} \le \max\{S_{i'}^{j'}\}$ **then**
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 k'th security service of task $T_{i'}$ will no longer be considered;
2.4 update the worst-case utilization $\sum_{i=1}^{n} \left[\left(e_i + \sum_{k-1}^{q} c_i^k (s_i^k) \right) \Big/ p_i \right]$;
end while

Figure 1. The SASES scheduling algorithm for periodic tasks.

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.

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.

$$\theta_{i'}^{k'} = \max_{1 \le i \le n, 1 \le k \le q} \left\{ \theta_i^k \right\}. \tag{10}$$

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

4. Experiment Results

We compare the proposed SASES with three baseline algorithms, which are variants of EDF (Earliest Deadline First). 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. (2) *maxsEDF*: The scheduler intentionally chooses the highest security level for each security requirement posed by each task. (3) *rndsEDF*: rndsEDF randomly picks a value within the security level range for each requested security service of a task. Table 2 summarizes the important configuration parameters of a simulated high-performance embedded system used in our experiments.

Table 2. Characteristics of System Parameters

Parameter	Value (Fixed) - (Varied)
CPU Speed	(100 million instructions/second or MIPS) – (100, 150, 200)
Task execution time	(100 ~ 1000) millisecond
Periods	(10 ~ 1000) millisecond
Mean size of data to be secured	(60, 90, 120) KB
Required security services	Encryption, Integrity, and Authentication
Weight of security service	(authentication: encryption: integrity) – (0.5, 0.3,0.2);(0.2,0.5,0.3); (0.2, 0.3, 0.5)

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 100MIPS. The period of each task is randomly generated in the range between 100 and 1000ms. Similarly, the worst-case computation time of each task is uniformly distributed between 10 and 1000ms. 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.

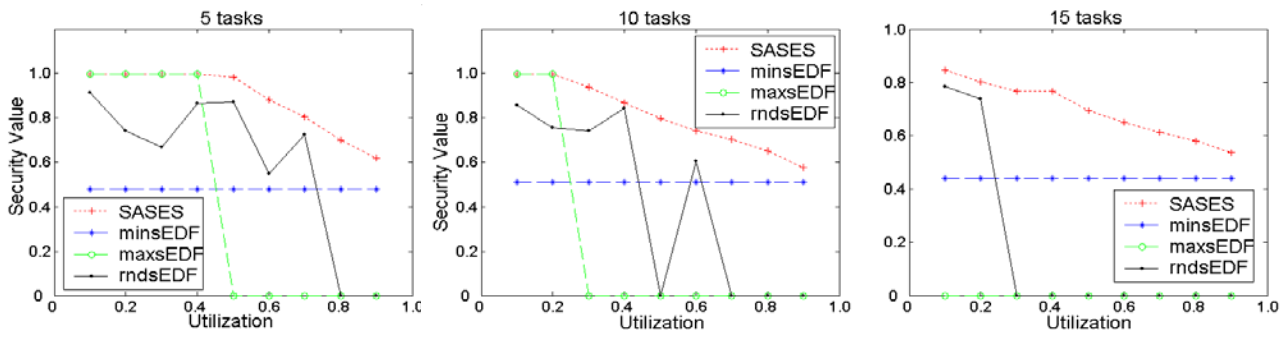

Figure 2. Performance impact of number of tasks.

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 high-performance embedded systems. These valuable characteristics show us that SASES can be successfully applied to real high-performance embedded systems.

5. Conclusion

This paper addressed the scheduling problem for periodic tasks with security and timing constraints. 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 high-performance 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 93.4% improvement in security.

In the future, we will develop a feedback control mechanism, which is enabled to dynamically adjust security levels for aperiodic tasks. 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

[1] T.F. Abdelzaher, E. M. Atkins, and K.G. Shin., "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control," *IEEE Trans. Computers*, Vol. 49, No. 11, Nov. 2000, pp.1170-1183.

[2] D. G. Feitelson, L. Rudolph, Job, "Scheduling for Parallel Supercomputers," *In Encyclopedia of Computer Science and Technology*, Vol. 38, New York, 1998.

[3] C.-J. Hou and K. G. Shin, "Allocation of Periodic Task Modules with Precedence and Deadline Constraints in Distributed Real-Time Systems," *IEEE Trans. Computers*, Vol. 46, No. 12, pp.1338-1356, Dec. 1997.

[4] M. Iverson and F. Ozguner, "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment," *In Proc. of the Seventh Heterogeneous Computing Workshop*, pp.70-78, Orlando, Florida, USA, 1998.

[5] D. Kebbal, E.G. Talbi, J.M. Geib, "Building and Scheduling Parallel Adaptive Applications in Heterogeneous Environments," *1st IEEE Computer Society International Workshop on Cluster Computing*, Melbourne, Australia, December 02 - 03, 1999.

[6] P. Koopman, "Embedded System Security," *IEEE Computer*, Vol. 37, No. 7, pp. 95-97, July 2004.

[7] A. Radulescu, Arjan J.C. van Gemund, "Fast and Effective Task Scheduling in Heterogeneous Systems," *In Proc. of the 12th Euromicro conferences on Real-time Systems*, pp229-238, 2000

[8] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in Embedded Systems: Design Challenges," *ACM Trans. Embedded Computing Systems*, Vol. 3, No. 3, pp. 461-491, Aug 2004.

[9] S. H. Son, R. Mukkamala, and R. David, "Integrating security and real-time requirements using covert channel capacity," *IEEE Trans. Knowledge and Data Engineering*, Vol. 12, No. 6, pp. 865 – 879, Nov.-Dec. 2000.

[10] M.E. Thomadakis and J.-C. Liu, "On the efficient scheduling of non-periodic tasks in hard real-time systems," *Proc. 20th IEEE Real-Time Systems Symp.*, pp.148-151, 1999.

[11] H. Topcuoglu, S. Hariri, M.-Y. Wu, "Task Scheduling Algorithms for Heterogeneous Processors," *In Proc. of 8th Heterogeneous Computing Workshop*, pp.3-14, 1999.

[12] T. Xie, X. Qin, and A. Sung, "SAREC: A Security-Aware Scheduling Strategy for Real-Time Applications on Clusters," *Proc. the 34th Int'l Conf. Parallel Processing*, Norway, June 14-17, 2005.