Microprocessors and Microsystems 52 (2017) 401–415

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Design optimization for security- and safety-critical distributed real-time applications

Wei Jiang[a,*], Paul Pop[b], Ke Jiang[c]

[a] School of Information and Software Engineering, University of Electronic Science and Technology of China, China
[b] Department of Compute, Technical University of Denmark, Denmark
[c] ÅF-Technology AB, Sweden

ARTICLE INFO

Article history:
Received 11 February 2016
Revised 23 July 2016
Accepted 9 August 2016
Available online 10 August 2016

Keywords:
Embedded system
Security
Safety
Energy
Design optimization

ABSTRACT

In this paper, we are interested in the design of real-time applications with security, safety, timing, and energy requirements. The applications are scheduled with cyclic scheduling, and are mapped on distributed heterogeneous architectures. Cryptographic services are deployed to satisfy security requirements on confidentiality of messages, task replication is used to enhance system reliability, and dynamic voltage and frequency scaling is used for energy efficiency of tasks. It is challenging to address these factors simultaneously, e.g., better security protections need more computing resources and consume more energy, while lower voltages and frequencies may impair schedulability and security, and also lead to reliability degradation. We introduce a vulnerability based method to quantify the security performance of communications on distributed systems. We then focus on determining the appropriate security measures for messages, the voltage and frequency levels for tasks, and the schedule tables such that the security and reliability requirements are satisfied, the application is schedulable, and the energy consumption is minimized. We propose a Tabu Search based metaheuristic to solve this problem. Extensive experiments and a real-life application are conducted to evaluate the proposed techniques.

1. Introduction

Embedded systems are increasingly used in critical areas, e.g., automotive and avionic electronic systems. Such systems are facing emerging challenges from security, reliability, energy, and timing requirements. In this paper, we refer to such systems as security- and safety-critical systems (SSCSs). SSCSs have to function correctly while meeting timing constraints even in the presence of faults. Such faults can be permanent, intermittent, or transient. In this paper, we focus on protecting the system against the most common type of faults, namely, the transient faults [1].

With the integration of new communication interfaces, SSCSs are exposed to increasingly severe security threats, hence the need of protecting sensitive communication information becomes of utmost importance [2]. The snooping, spoofing, or alteration of critical data does not only compromise security but can also lead to system failure, resulting in great financial loss and potentially endangering human life and the environment. For example, disclosure or tampering of critical messages, e.g., of braking or acceleration, in automotive electronic systems can compromise vehicles security and, consequently, safety. Although embedded system security has been addressed in literature [3,4], security issues in distributed embedded system communication, especially the internal communication, have not received as much attention. In [5], several attack scenarios in automotive networks were discussed, and the importance of utilizing cryptography to protect the internal bus communication was highlighted. To provide system-affordable security protection for the internal communication of distributed SSCSs, it is of critical importance to find efficient solutions at early design stages [6,7]. Considering the security issues on mixed-criticality real-time applications, authors of [8] proposed a GA (Genetic Algorithm) based efficient heuristic algorithm to address the system-level optimization of the security-sensitive mixed-criticality real-time applications. However, the underlying energy and reliability issues in the context of distributed SSCSs were not addressed in these works.

Energy efficiency is a fundamental requirement of many embedded systems, not only in battery powered systems. In SSCSs, quick energy depletion or early exhaustion of batteries may compromise security and even cause failure of mission-critical tasks, resulting in unexpected outcomes. One of the most common approaches in energy management is to utilize dynamic voltage

* Corresponding author.
E-mail addresses: weijiang@uestc.edu.cn (W. Jiang), paupop@dtu.dk (P. Pop), dalingog@gmail.com (K. Jiang).

http://dx.doi.org/10.1016/j.micpro.2016.08.002
0141-9331/

& frequency scaling (DVFS) [9]. The effectiveness of DVFS for the modern processors is debatable because of their reduced dynamic power range [10]. However, safety-critical systems typically use older-generation architectures (for which detailed failure data is available). DVFS can have a negative impact on the system reliability, since lowering the voltage leads to exponential increase in the number of transient faults as shown in [11]. Based on such DVFS-related reliability model, efficient reliability management mechanisms have been presented for energy-critical uni-processor systems and distributed systems [12,13]. However, the security issues were seriously overlooked in these works.

In this paper, we are interested in optimizing the energy consumption for security- and reliability-critical applications on DVFS-enabled heterogeneous distributed real-time embedded systems, where both tasks and messages are statically scheduled. We consider both security and reliability issues together for DVFS-enabled real-time embedded systems with a general application model, which is highly different to the work on energy optimization of security-related mixed-criticality real-time applications [8]. Due to the huge complexity of the problem, we propose an efficient Tabu Search-based heuristic that decides on the system variables, i.e., the voltage levels and start time of tasks, the transmission time of messages and their security levels, such that the energy consumption is minimized, while the reliability, real-time, and security requirements of the application are satisfied. The main contributions of this paper are: (1) we introduce a vulnerability based method to quantify the security performance of communications on distributed systems; (2) we address a unified design problem for security- and safety-critical embedded systems which simultaneously considers the security, reliability, energy, and timing requirements; (3) we evaluate the proposed approach on several synthetic benchmarks and one real-life case study (a vehicle cruise controller application) and compare it to other approaches from related work.

The rest of this paper is organized as follows. Section 2 describes the system application and hardware architecture. Sections 3 and 4 present the security model and DVFS-related reliability model we used in this paper, respectively. Section 5 formulates the design optimization problem, and depicts an illustrative example. Our TS-based heuristic and experimental results are given in Sections 6 and 7, respectively. The conclusions of the paper are drawn in Section 8.

2. System architecture

In this section, we describe both application model and hardware architecture for SSCSs. For future reference, we give the most used symbols and abbreviations of this paper in Table 1.

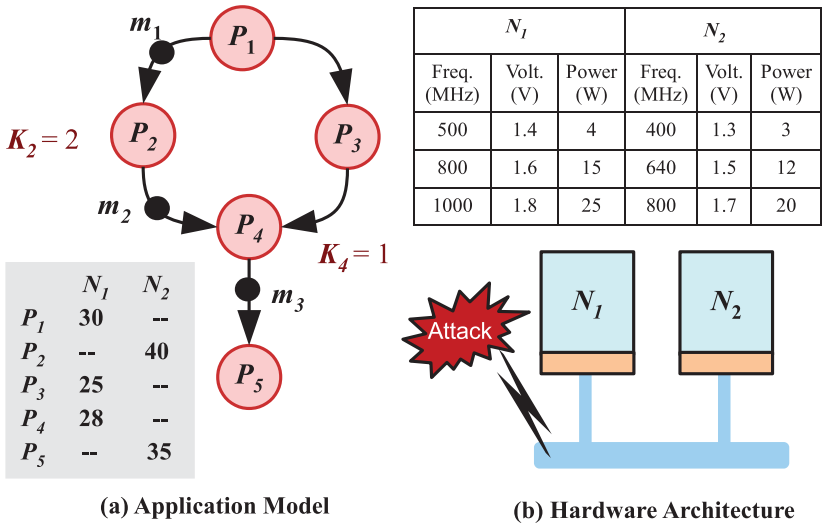
2.1. Application model

We model the application as a directed acyclic task graph $G(\mathbb{V}, \mathbb{E}, \mathbb{M})$. $\mathbb{V}$ is the set of vertices, and each node in $\mathbb{V}$ represents one task P_i. An edge $e_{ij} \in \mathbb{E}$ from P_i to P_j indicates that there is a data dependency between P_i and P_j. $\mathbb{M}$ is the set of messages that need to be transmitted over the communication bus. We assume that the mapping of tasks to processors is given by the designer based on his/her former experience, depending on the concrete execution requirements of the application and usage constraints. The worst case execution time (WCET) C_i of each task P_i is known. In this paper, we focus on the confidential security of messages delivered over the internal communication bus of distributed systems. Thus, messages exchanged by tasks on the same processor are assumed to be secure (e.g., protected by memory isolation, with no communication bus delivery), and their transmission time is ignored. These messages are not explicitly modelled in our application graph. Thus, we only model the messages between tasks on different processors, which are transmitted over the internal communication network. Such messages are denoted with a solid circle on top of the corresponding edges. A task is activated when all its inputs have arrived. Tasks are non-preemptive, and thus, cannot be interrupted during execution. Fig. 1(a) depicts an application consisting of five tasks, P_1 to P_5, and two communication messages, m_1 and m_2, which are supposed to be executed on two processors (N_1, N_2). The mapping and the WCETs (in μs, considering the highest operating mode) are also given in the grey table of Fig. 1(a). The application must finish all executions within D time units after released, known as its global deadline.

We concentrate on tolerating the transient faults of the tasks. Several fault-tolerant techniques have been proposed for this purpose [14], e.g., re-execution, replication and checkpointing with rollback recovery. The techniques have different advantages and disadvantages in terms of performance and energy consumption. In this paper, we have chosen to use active task replication for its simplicity in terms of scheduling. Note that our proposed techniques can be easily extended to support other fault-tolerant techniques. The designer specifies, for each critical task P_i, a number of K_i replicas to ensure execution reliability. For example in Fig. 1(a), task P_2 has two replicas, i.e., $K_2 = 2$. We assume that error detection is performed by each critical task and its replicas. In case a majority voting on the outputs is required, we assume that the designer will add such a voter task to the application graph. The mapping of these replicas is also given by the designer and can be on a different or on the same processor with P_i.

Table 1
Mostly used symbols and abbreviations.

Notation	Definition
G	Application graph
$\mathbb{V}$	The set of tasks
$\mathbb{M}$	The set of messages
$\mathbb{E}$	The set of dependency relations of tasks
$\mathbb{N}$	The set of DVFS-enabled processing nodes
P_i	The i-th task
N_i	The i-th processing node
m_i	The i-th message
D	Deadline of application G
C_i	Execution time of P_i
L_i	Security level of m_i
K_i	The number of replicas of P_i
SI_i	Security input task of m_i
SO_i	Security output task of m_i
RE	System reliability
RE_B	System reliability bound
En	System energy consumption
OV	Overall vulnerability
VD	Vulnerability deficiency
MOV	Maximal overall vulnerability
OV_B	The upper bound of overall vulnerability
VB_i	The vulnerability bound of m_i
η	System reliability degradation
Φ	DVFS mode assignment for all tasks
Υ	Security level assignment for all messages
x	The solution for the optimization problem
LS	List scheduling
TS	Tabu search
DVFS	Dynamic voltage & frequency scaling
SSCSs	Security- and safety-critical systems
TSHO	Tabu search-based hybrid optimization
GHO	Greedy-based hybrid optimizing
MVFS	Mapping, voltage and frequency scaling optimization
EO	Energy optimization
RSO	Reliability and security constrained optimization
VCC	Vehicle cruiser controller
WCET	Worst case execution time
CP	Critical path
MPCP	Modified partial critical path



N_1			N_2		
Freq. (MHz)	Volt. (V)	Power (W)	Freq. (MHz)	Volt. (V)	Power (W)
500	1.4	4	400	1.3	3
800	1.6	15	640	1.5	12
1000	1.8	25	800	1.7	20

Fig. 1. System architecture.

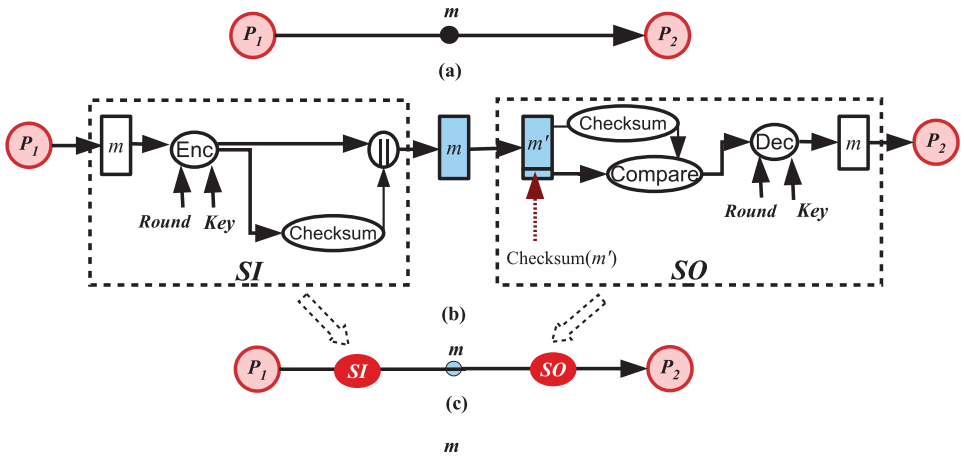


Fig. 2. Critical message security.

2.2. Hardware architecture

The target hardware architecture is a distributed embedded system composed of $\mathbb{N}$ DVFS-enabled processing nodes (as in [15]) that are connected by an internal communication bus as illustrated in Fig. 1(b). A processor is characterized by a set of discrete operating modes. Each mode, $\phi_i^{N_j} = (f_i^{N_j}, v_i^{N_j}, pw_i^{N_j})$, of node N_j consists of three parameters: $f_i^{N_j}$ is the operating clock frequency; $v_i^{N_j}$ is the supply voltage; $pw_i^{N_j}$ is the power. For simplicity, we denote $F_i^{N_j} = f_i^{N_j}/f_{max}^{N_j}$ and $V_i^{N_j} = v_i^{N_j}/v_{max}^{N_j}$ as the normalized frequency and voltage, respectively. Fig. 1(b) shows two processors, each with three operating modes.

3. Security model

3.1. Security overhead of messages

In this paper, we focus on protecting confidentiality of each message communication by block ciphers (BCs), e.g., 3DES, IDEA, RC5, RC6, SKIPJACK, and Blowfish. Let us consider the example in Fig. 2. P_1 and P_2 are two tasks exchanging a message m. Assuming that P_1 and P_2 are mapped to different processing nodes, m will then be transferred over the internal communication bus, and must be protected against eavesdropping attack, i.e., we must ensure its confidentiality. *SI* and *SO* denote the Security Input task (before message transmission) and Security Output task (after message reception), respectively. *SI* includes message encryption and checksum generation, while *SO* includes checksum verification and message decryption. Note that, *SO* is the inverse procedure of *SI*. Hence, *SI* and *SO* have the same execution time if executed with the same hardware configuration, which is also evaluated by the measurement on our testbed. In this paper we consider the widely used RC6 [6], notable for its simplicity and flexibility, for the encryption and decryption of sensitive messages.

Table 2
Security comparison of RC6 variants.

Level	♯*Rounds*	♯*Plaintexts*	*Vul*	θ (μs/Block)
1	4	2^{29}	98	3.08
2	8	2^{61}	67	3.58
3	12	2^{94}	34	4.15
4	16	2^{118}	10	4.63
5	20	2^{128}	0	5.21

We use *Security Level* to capture the relative strengths of confidentiality techniques used for a message. The security level of a cryptographic algorithm is reflected by its robustness against attacks. Taking RC6 as a example, the relative security level is directly affected by the number of encryption rounds as shown in the *level* and ♯*Rounds* columns of Table 2. However, the designer can find other tradeoff tables of robustness and throughput based

on the relative merits of employed services and his own experience. The approach presented in this paper can take as input any rankings provided by the designer. ♯*Plaintexts*, *Vul* and θ in Table 2 present the number of plaintexts needed for successful cryptanalysis attack, the vulnerability metric (see the next subsection) and the encryption/decryption overhead for RC6 variants,[1] respectively.

Note that we must use the same security algorithm on the sender and receiver sides. Therefore, the task that generates and receives the message will have the same security level. Similar to [3], the security related execution time on message m_i can be characterized by a linear function of data size and selected security level as follows,

$$C_i^{SI} = C_i^{SO} = I_i + \theta(L_i) \cdot S_i. \tag{1}$$

I_i is the WCET for doing the pre-/post-whitening on the encryption and decryption. In this paper, we also incorporate the overhead of checksum generation or verification within I_i. L_i is the security level for message m_i, which indicates the algorithm selection of confidentiality. S_i is the size of the message in blocks. $\theta(L_i)$ is the worst-case computation time of the BC function for one block (128 bits) of data on level L_i. C_i^{SI} and C_i^{SO} are the WCETs of *SI* and *SO* at the maximal frequency on the fastest processor. If the hardware of the processing nodes executing *SI* and *SO* is different, then $C_i^{SI} \neq C_i^{SO}$.

We have measured the execution overheads of RC6 variants based on a real-time operating system, µC/OS-II, running on a hardware platform with a S3C2440 ARM processor (500 MHz) and 64 MB SDRAM. The measurement instrument is PXI 1024Q with a NI 6221 data acquisition card [16]. More information about the testbed can be found in [17].

RC6 is a parametrized family of symmetric algorithms. Each variant of RC6 is accurately specified as RC6-w/r/b, where w is the word size in bits, r is the number of encryption rounds and b denotes the key size in bytes. We focus in this paper on the impact of encryption rounds, then we set the word size as 32 bits, and the key size of RC6 as 16 Bytes (128 bits). This also means that we deploy the operating mode of RC6 as RC6-32/r/16 in the measurement, and we only reconfigure r to change the operating mode and test the corresponding overhead. Note that the time overhead is independent of key size for RC6 according to our measurement results. We set the data size (block size) as 16 Bytes (128 bits). The measured WCETs (per unit of data block) of different security levels on our testbed, $\theta(L_i)$, are shown in the last column of Table 2. In the measurement, we also obtain that the encryption overhead is equal to the decryption overhead for the same security level.

3.2. Security quantification and constraints

We assume that the system applies RC6 to protect confidentiality of message communications. RC6 is a iterated block cipher, notable for its simplicity and flexibility. With a variable number of rounds, the user can explicitly manipulate the trade-off between higher throughput and higher security. All the follow up discussions are based on the assumption of applying RC6 in the system. But our techniques is also applicable to other iterated block ciphers.

One effective way to quantify the security is to compare the vulnerability of a cipher to a cryptanalysis attack against an exhaustive key search [18]. There have been a lot of cryptanalysis evolutions since the publication of RC6. For example, references [19,20] have analysed the security performance of RC6 against linear and differential cryptanalytic attacks. Cryptanalytic attacks are theoretical attacks, and require a large number of plaintexts to successfully estimate the encryption key bit(s). Required amount of such plaintexts grows exponentially as encryption rounds increase. The number of rounds (from 4 to 16) and corresponding required plaintexts by the linear cryptanalytic attack for key recovery are presented in Table 2. RC6 with 20 rounds is considered to be secure enough against cryptanalytic attacks, since the amount of required plaintexts exceeds 2^{128} plaintexts [19]. Consequently, the brute force attack is more effective, which requires maximally 2^{128} plaintexts. Thereby, 2^{128} is listed as last item of the third column in Table 2.

In this paper, we use a vulnerability metric as the quantification of each message's security strength. Similar to [18], we define the vulnerability, V_i, as the logarithm of the ratio of the maximum number of plaintexts required by the brute-force search, PT_{bf}, to the number of plaintexts required using a chosen cryptanalysis algorithm, PT_{cc}.

$$V_i = log_2\left[\frac{PT_{bf}}{PT_{cc}(L_i)}\right] \tag{2}$$

$PT_{cc}(L_i)$ is the required plaintexts to successfully attack RC6 variant with security level L_i by linear cryptanalytic method. The number of plaintexts needed by brute-force is equal to the power of block size, for example attacking RC6 (block size 128 bits) requires $PT_{bf} = 2^{128}$ plaintexts. Based on the case of linear cryptanalysis, the vulnerabilities of RC6 operating in these five chosen variants are calculated and listed as the 4-th column in Table 2. For example, a successful attack on RC6 with 12 rounds (security level 3) requires 2^{94} plaintexts, then the vulnerability is $log_2(2^{128}/2^{94}) = 34$.

We define the system vulnerability as the Overall Vulnerability (*OV*) of all critical messages. We assume that the designer will specify a system vulnerability bound OV_B for the whole security-critical application. In this paper we assume the weight of each message's vulnerability is different due to the sensitivity of the message content. Denoting w_i as the weight of m_i, we have the system vulnerability constraint as follows,

$$OV(\mathbb{M}) = \sum_{m_i \in \mathbb{M}} w_i \cdot V_i \leq OV_B. \tag{3}$$

The system vulnerability bound is intended to reduce the overall system vulnerability by protecting the messages. However, there may be messages that are highly security-critical, which if decrypted would jeopardise the system security. Our assumption is that, similar to the safety-critical tasks for which the designer assigns a number of replicas to increase their reliability, the designer will identify the most security-critical messages, and specify for each of them an individual vulnerability bound VB_i. By reducing the vulnerability of highly security-critical messages, together with the overall system vulnerability, we guarantee that there are no "weakest link" messages that can be attacked to jeopardise the security.

3.2.1. Short discussion

In this paper, we focus on the confidentiality protection of messages. We take RC6, one widely used confidential algorithm, as an example to protect the messages. Although the main work of this paper is based on the assumption of applying RC6 in the system. But our techniques are also applicable to other confidential algorithm (iterated block ciphers), e.g., RC5 and AES [18]. Generally speaking, there are three types of security protections (i.e., authentication, integrity and confidentiality) need to be considered for distributed embedded systems to resist attacks of snooping, alteration and spoofing [3]. To incorporate these security protections into our model, we can extend our model by considering an accumulated weighted vulnerability model similar to [3]. For example, if message m_i needs to be protected by both confidentiality

[1] We denote the cipher with a concrete encryption/decryption round as one of its variants.

and integrity protection, we can use $V_i = w_i^{conf.} V_i^{conf.} + w_i^{int.} V_i^{int.}$ to model the whole vulnerability. Our vulnerability for confidential protection cannot be directed used for other type of security protections, but we can use similar method to model the corresponding vulnerability by attack complexity analysis, e.g., the complexity of collision attack on the family of SHA algorithms for integrity protection.

Note that, in this paper we only focus on the message confidential protection over the internal communication bus, while we ignore the protection operations on the processor during the encryption or decryption of messages. We assume the system designer has deployed high-level security protection on the processors, which can resist Side Channel Attacks. Along with the communication security protection, considering security protections to resist Side Channel Attacks in the system-level will be a very interesting work, which will be conducted in our future work.

4. Energy/reliability model

Although DVFS is a practical approach for reducing the energy consumption of embedded systems, it can have a negative impact on reliability. Recent works have shown that decreasing the processor frequency and voltage will increase the transient fault rate [11,12]. The fault rate is influenced by the operation mode, i.e., the supply voltage and frequency, in which the processor runs. Given the operation mode $\phi_i^{N_j}$ with normalized frequency $F_i^{N_j}$ and voltage $V_i^{N_j}$, the fault rate is calculated as in [13], $\lambda(\phi_i^{N_j}) = \lambda_0 \cdot (F_i^{N_j})^a \cdot 10^{-bV_i^{N_j}}$, where a and b are frequency and voltage related architecture dependent constants, and λ_0 is the fault rate when system runs with maximal voltage $v_{max}^{N_j}$ and frequency $f_{max}^{N_j}$.

The reliability RE_i of task P_i under operating mode $\phi_i^{N_j}$ is the probability of its successful execution. This is captured using the exponential failure law,

$$RE_i = exp\left(-\frac{\lambda(\phi_i^{N_j}) \cdot C_i}{F_i^{N_j}}\right), \tag{4}$$

where C_i is the WCET of task P_i under the highest operation mode $\phi_{max}^{N_j}$ with frequency $f_{max}^{N_j}$. If the critical task P_i needs to be replicated K_i times, then its overall reliability can be calculated as follows,

$$RE_i^{critical} = 1 - \prod_{k=1}^{K_i+1}\left(1 - exp\left(-\frac{\lambda(\phi_{i,k}^{N_j}) \cdot C_i}{F_{i,k}^{N_j}}\right)\right), \tag{5}$$

where $F_{i,k}^{N_j}$ and $\phi_{i,k}^{N_j}$ indicate the frequency and operating mode of the k-th replica of P_i, respectively.

Not all tasks in an application are critical. Let us assume that there are h normal tasks and g critical tasks, then the reliability of whole application is calculated as:

$$RE = \prod_{i=1}^{h} RE_i \cdot \prod_{i=1}^{g} RE_i^{critical} \tag{6}$$

We assume that the designer will specify a system reliability bound RE_B. The idea is that if the system reliability is below this threshold, then more replicas would be needed to tolerate the failures, i.e., the system is no longer fault-tolerant. Hence, we impose the reliability constraint as $RE \geq RE_B$.

5. Design problem and motivational example

5.1. Problem formulation

In this paper, we are interested in minimizing the energy consumption of security- and safety-critical real-time applications on DVFS-enabled distributed systems. Based on the application model of [21], we also assume that all collaborative tasks and messages belonging to an application graph G have the same period, which is the period of the application graph. We have a global deadline constraint D on the end-to-end completion time, *FinishTime*, of the application G, which is less than its period. Applications can have associated individual release times and deadlines. Multiple-rate applications can also be handled by merging them into a single graph G with the hyper-period, i.e, the least common multiple, of the application periods. The security and reliability requirements have been described in Sections 3 and 4. For simplicity, we assume that (1) the message communication is reliable, i.e., we will not consider fault-tolerance for messages, and (2) the *SI* and *SO* tasks use the highest operating mode due to their importance.

We focus on the energy management of processing nodes, which is the main component of the overall system energy. But our model can be extended to capture also the energy consumption by leakage, memories, I/Os, etc. The energy consumption of the application is composed of two parts, namely energy consumed by task executions and message security protection operations. For each task P_i running in the operating mode $\phi_l^{N_j}$, the energy consumption is:

$$En(P_i) = C_i \cdot pw_l^{N_j} / F_l^{N_j} \tag{7}$$

For security protections of each message m_i, energy consumption is:

$$En(m_i) = C_i^{SI} \cdot pw_l^{N_j} / F_l^{N_j} + C_i^{SO} \cdot pw_q^{N_k} / F_q^{N_k} \tag{8}$$

where $pw_l^{N_j}$ and $pw_q^{N_k}$ are the power in the operation mode $\phi_l^{N_j}$ and $\phi_q^{N_k}$ for the *SI* and *SO* tasks, respectively.

The energy consumption and execution time of the application depend on the operating mode assignment Φ and the security level assignment Υ. The system reliability is also related to operating mode assignment Φ, and the system vulnerability relies on the security assignment Υ.

The problem we address in this paper can be formulated as follows. Given an application G and its mapping to an architecture of $\mathbb{N}$ processing nodes, each with a set of operating modes ϕ and a set of chosen security services (Table 2), we are interested to determine an implementation solution x such that the energy is minimized, and the requirements on schedulability (deadline D), security (system vulnerability bound OV_B and individual vulnerability VB_i of each critical message), and reliability (RE_B) are satisfied. Deriving an implementation solution x means deciding on the set of operating modes Φ for the tasks, the set of security levels Υ for the messages and the scheduling table Γ for both tasks and messages. The problem can be formally expressed as:

$$\text{Min } En = \sum_{P_i \in \mathbb{V}} En(P_i) + \sum_{m_i \in \mathbb{M}} En(m_i) \tag{9}$$

Subject to

$$RE \geq RE_B \tag{10}$$

$$FinishTime(G) \leq D \tag{11}$$

$$OV(\mathbb{M}) \leq OV_B \text{ and } V_i \leq VB_i, \forall m_i \in \mathbb{M} \tag{12}$$

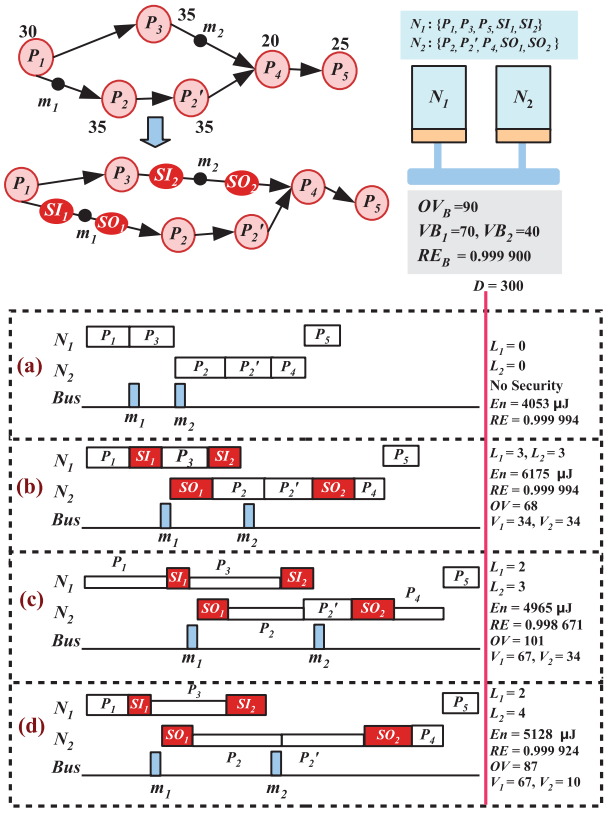


Fig. 3. Motivational example.

5.2. Motivational example

An illustrative example is depicted in Fig. 3. The application consists of five tasks mapped to a DVFS-enabled architecture of two processing nodes connected by a bus. We use the same hardware architecture as Fig. 1(b). The task mapping and corresponding task parameters can be found in the figure. Let us assume that P_2 is a critical task for which the designer has introduced one replica, P_2', to tolerate one transient failure, and mapped the replica on the same processing node, N_2. The reliability goal is set to be $RE_B = 0.9999$, which is a little lower than upper bound reliability (0.999994), assigning all tasks with the fastest frequency. The system vulnerability and individual vulnerability demands of messages m_1 and m_2 are $OV_B = 90$, $VB_1 = 70$ and $VB_2 = 40$, respectively. The application end-to-end deadline is set as $D = 300$ time units. We use the vulnerability value of each security level according to Table 2. For simplicity, we assume the vulnerability weight of both messages are the same, i.e., $w_1 = w_2 = 1$.

Without considering the security requirements and DVFS, we can get the shortest schedule of the application as in Fig. 3(a), in which the deadline constraint (depicted by a red vertical line) and the reliability goal RE_B of the solution (shown to the top of the schedule) are both satisfied. The energy consumption is $En^a = 4053$ µJ, and the end-to-end delay is $FinishTime(G)^a = 180$. However, there is absolutely no security protections achieved at this moment.

One way to meet the security requirements is to set the security levels for messages m_1 and m_2 to be $L_1^b = L_2^b = 3$, which satisfies the security requirements, i.e., $OV = 68 < OV_B$, and $V_1 = 34 < VB_1, V_1 = 34 < VB_1$. The resulted schedule, which takes into account the security input (SI) and security output (SO) tasks for each message is depicted in Fig. 3(b). The WCETs for *SI* and *SO* tasks are determined as discussed in Section 3, and depend on security level L_i. The energy consumption in this case is $En^b = 6175$ µJ, and the schedule length is $FinishTime(G)^b = 215 < D$. The system reliability is the same as in Fig. 3(a), and meets the bound RE_B.

Our ultimate goal is to minimize energy consumption of the application. With DVFS, we can scale down the operating mode for tasks to save energy, e.g., as the case in Fig. 3(c). To give more flexibility to DVFS, we lower the security level of m_1 to $L_1^c = 2$. Processes P_1, P_2, P_3, and P_4 are set to run at the lowest operating mode on their processor, while operating modes of other tasks are not changed. By this, the energy is reduced by 19.6% from En^b. However, the system vulnerability $V_1^c + V_2^c = 101$ violates the requirement $OV_B = 90$ even though the individual vulnerability requirement is satisfied, i.e., $V_1 = 67 < VB_1, V_1 = 34 < VB_1$. The reliability bound is also violated as $RE^c = 0.998671 < RE_B$, since lowering operating mode (the frequency and voltage) increases the failure rate.

However, if we do operating mode assignments for tasks and security level selections for messages more carefully, it is possible to meet the timing, security, and reliability requirements without sacrificing energy savings. Fig. 3(d) depicts the optimal solution that can be reached. We choose to change the security levels to be $L_1^d = 2$ and $L_2^d = 4$. The system vulnerability requirement $OV_B = 90$ and the individual vulnerability demands of m_1 and m_2 are both guaranteed. In addition, P_3 is scaled to use the second operating mode with frequency 800 MHz on N_1. The operating modes of P_2 and P_2' are all scaled down to the lowest operating mode on N_2. P_1, P_4 and P_5 are changed to highest operating modes. In this case, we can save energy by 17.0% from En^b, which is comparable to the third scenario (Fig. 3(c)). Meanwhile, all operational constraints are satisfied. This shows that we need to carefully optimize the assignment of operating modes and security levels in order to meet the imposed requirements and minimize the energy consumption. The proposed solution to this optimization problem is presented in the next section.

6. Design optimization strategy

6.1. Design optimization strategy overview

Due to the huge complexity of the problem, we choose Tabu search as our optimization approach. Tabu search is a very efficient neighbourhood search, considering the most recently visited solutions as tabus to avoid the local optimum. Tabu has been widely used for design optimization of embedded systems like in [14,22].

We propose a Tabu Search-based Hybrid Optimization (TSHO) framework to search for the best solutions. Tabu Search (TS) is used to decide the mode assignment Φ for the tasks and the security level assignment Υ for the messages. We use List Scheduling (LS) policy [23] to determine, for a given Φ and Υ, the schedule table Γ. The TSHO design optimization strategy is carried out in five consecutive steps as shown in Algorithm 1.

Algorithm 1 Tabu search-based hybrid optimization (TSHO)

Input: Application G, Hardware arch. $\mathbb{N}$
Output: x^*, En, OV, RE

1: $SecExtend(G, \mathbb{N})$ /*Add security-related tasks into the system*/
2: $x^0 = (\Phi^0, \Upsilon^0)$ /*Generate the initial solution*/
3: $x^* = TS(x^0)$ /*Tabu Search for the best solution $x^* = (\Phi^*, \Upsilon^*)$*/
4: $\Gamma^* = LS(G, \mathbb{N}, \Phi^*, \Upsilon^*)$ /*Build the schedule table using extended List Scheduling*/
5: Calculate energy En, vulnerability OV and reliability RE based on x^*

In the first step of Algorithm 1, we extend the application with security related tasks, namely *SI* and *SO*, that are mapped to the same processing nodes as messages sender and receiver tasks, respectively. In the second step, an initial solution $x^0 = (\Phi^0, \Upsilon^0)$ is generated with lowest security level (that can guarantee the individual vulnerability requirement) for each message and highest operating mode for each task. Given the input x^0, TS searches for the best solution x^* which minimizes the energy consumption under the constraints in the third step. In the fourth step, we extend the basic LS policy to determine the schedulable table Γ^* for the application based on the solution $x^* = (\Phi^*, \Upsilon^*)$ returned by TS. We finally calculate the energy consumption, overall vulnerability and reliability of the application in Step 5.

The cost function which TS uses to evaluate each solution x is as follows,

$$cost(x) = En + \sum_{e \in (D,RE,OV)} W_e \cdot degree(e). \tag{13}$$

The first term is the energy consumption En to be minimized and the second term is used to check if the timing D, reliability RE_B and system vulnerability OV_B requirements are satisfied. Instead of neglecting solutions which violate constraints, we penalize constraint violations in the cost function based on the user defined penalty weights of W_D, W_{RE} and W_{OV}. By this, we can explore even the infeasible regions of the search space in order to find better quality solutions. All these requirements are defined in a similar way using a "degree function", which checks the degree to which the requirement is met. If the constraint is satisfied, then $degree(e) = 0$. Otherwise, it is how much the constraint is violated. For example, the system vulnerability constraint $degree(OV)$ is defined as follows,

$$degree(OV) = \begin{cases} 0, \text{ if } OV(\mathbb{M}) \leqslant OV_B \\ OV(\mathbb{M}) - OV_B, \text{otherwise.} \end{cases}$$

Similarly, we can calculate the degree of reliability as $degree(RE) = RE_B - RE$ in case of reliability violation. For each solution TS visited, we use LS to determine the finish time of the application G, then we calculate $degree(D) = FinishTime(G) - D$, if the deadline is violated.

Note that the purpose of incorporating penalty function in the base of cost function is to explore even the infeasible regions of the search space, so as to find better quality solutions. We set penalty weights of W_D, W_{RE} and W_{OV} according to user requirement. If user cannot tolerate the solutions violating the constraints, a very large corresponding weight will be expected to be set. For example, we can set high value to W_D for hard real-time applications. This will result in a high-cost solution when deadline is violated, and the generated solutions will not be chosen for next searching iterations. These weights only have the impact on the selection of intermittent solutions. These weights are insensitive to the final solution. This is because lots of solutions without violating all these constraints (having much lower cost than punished solutions) can be found to generate the final solution by TSHO. The most important role of penalty function is to enhance the diversification of TSHO during the searching procedure.

6.2. Tabu search

In this section, we focus on the Tabu Search optimization, which is shown in Algorithm 2. TS is a meta-heuristic which searches for the best solution with respect to the cost function. Starting from a current solution, TS uses design transformations (called moves) to generate neighbouring solutions. To escape from a local optimum, TS incorporates an adaptive memory (Tabu list) to prevent the search from revisiting previous solutions, thus avoiding cycling. In each iteration, we define the qualified candidate set as non-tabu solutions (generated by moves, but not in the Tabu list), and the "aspirated" solutions (better than the best-so-far solution, regardless of their tabu status). TS uses "diversification" by randomly selecting the older moves in Tabu list to direct the search into unexplored region. To find the best solution in the current iteration, we prefer a solution in the qualified candidate set that is better than the best-so-far solution. If such a solution does not exist, then we choose the best one in the diversification candidate set. If no diversification solution exits, we simply set current solution as the best non-Tabu solution in this iteration. We choose the final best solution x^* as the current solution if its cost is globally better.

We develop the TS procedure as a two-level tabu searching framework, which is composed of security level searching and operating mode searching. We define two types of moves: Security Moves (SecMove) and Operating mode Moves (OpMove). After the initialization (lines 1–2), we firstly use TS to generate a set of security solutions by SecMove (line 4). For each security solution, we search for the best operating mode solution (lines 5–13), then in the outer iteration we can focus on searching for the best security solution (lines 14–19). For each kind of move, we utilize two operations to generate neighbouring solutions: ± 1 and *swap* (lines 4 and 6). In a ± 1 operation, TS increases or decreases the operating mode or security level of a solution with one step. A *swap* operation within OpMove means to swap the operating modes of two selected tasks, whereas for a SecMove, it means to swap the security levels of two messages. Since evaluating all neighbouring solutions can be extremely expensive for large applications, we design these moves to be stochastic. For example, in an OpMove, we choose each task with a certain probability to do ± 1 operation and do the *swap* operation for two randomly selected tasks. We take non-improving search for a certain number

Algorithm 2 Tabu Search (TS)

Input: x^0, $SecExtend(G, \mathbb{N})$, RE_B, D, OV_B
Output: x^*

1: $x_{best} = (\Phi^{best} = \Phi^o, \Upsilon^{best} = \Upsilon^o)$; $bestCost = cost(G, \mathbb{N}, \Lambda, x^o)$
2: $Tabu^{\Phi} = \varnothing$; $Tabu^{\Upsilon} = \varnothing$ /*Init Tabu lists of OpMove and SecMove*/
3: **while** not reach the stop condition **do**
4: $\Omega = SecMoves(\Upsilon^{best} : \pm 1, swap)$
5: **for** each $L \in \Omega$ **do**
6: $\Omega' = OpMove(\Phi^{best} : \pm 1, swap)$
7: /*determine qualified candidate set by adding non-tabu and aspirated solutions*/
8: $\Omega'_{qlf} = \{\Phi \notin Tabu^{\Phi} \quad \text{or} \quad aspirated(\Phi) \in Tabu^{\Phi} | \Phi \in \Omega'\}$
9: /*add diversification candidates*/
10: $\Omega'_{div} = \{$ choose $\Phi \in Tabu^{\Phi}$ with probability p $\}$
11: /*for each SecMove, we find the best OpMove*/
12: Find $\Phi^{best} \in \Omega'_{qlf} \cup \Omega'_{div}$
13: $Update(Tabu^{\Phi}), execute(\Phi^{best})$ /*Renew the tabu list and perform this OpMove*/
14: $\Omega_{qlf} = \{\Upsilon \notin Tabu^{\Upsilon} \quad \text{or} \quad aspirated(\Upsilon) \in Tabu^{\Upsilon} | \Upsilon \in \Omega\}$ /*the qualified set*/
15: $\Omega_{div} = \{$ choose $\Upsilon \in Tabu^{\Upsilon}$ with probability p $\}$ /*add diversified candidates*/
16: Find $\Upsilon^{best} \in \Omega_{qlf} \cup \Omega_{div}$
17: $Update(Tabu^{\Upsilon}), execute(\Upsilon^{best})$ /*Renew the tabu list and perform temporarily best SecMove*/
18: **if** $cost(\Phi^{best}, \Upsilon^{best}) < bestCost$ **then**
19: $x^* = (\Phi^{best}, \Upsilon^{best})$, $bestCost = cost(\Phi^{best}, \Upsilon^{best})$
20: Return $x^* = (\Phi^*, \Upsilon^*)$

of iterations as the stop condition. Then TS terminates and returns the best-found-solution x^* (line 20).

6.3. Extended list scheduling

In this paper, we use LS policy to determine the schedule tables for the application. LS utilizes a priority list of ready tasks, L_{ready}, and schedules the ready task with the highest priority. A task P_i is scheduled when it is ready (its predecessors have finished, and its messages have arrived), and it has the highest priority in the ready list L_{ready}. We extended the classical LS to take into account also the tasks reliability and messages security operations. Considering the bus as a computation node for messages, we firstly extend $L_{ready} = L_{ready}^{task} \cup L_{ready}^{Msg}$. Then we assign a Critical Path (*CP*) priority value to both *SI, SO* tasks and messages. We sort the ready list using the Modified Partial Critical Path (MPCP) priority function [23], in which dependent tasks have the same priority (*CP* value) if assigned to the same processing node. Thus, the priority of *SI* and *SO* tasks is set to the *CP* values of message's original predecessor and successor tasks, respectively. The *CP* of each message is equal to the sum of the *CP* of its successor task and the communication time needed for bus transmission.

With the extended List Scheduling policy, we can check whether the deadline of each temporarily generated solution is violated in the TS procedure, and determine the application scheduling table when the best-so-far solution is found.

7. Experimental evaluation

We conducted extensive experiments to evaluate the efficiency of our proposed TSHO framework. All experiments were performed on a Windows desktop machine with a dual-core CPU at 2.5 GHz and 4 GB of RAM. TSHO was implemented in C#. For comparison purposes, we also compared TSHO with four other approaches, namely MVFS, GHO, EO and RSO.

- MVFS (Mapping, Voltage and Frequency Scaling optimization): A reliability-aware mapping and DVFS optimization algorithm proposed in [13], aiming for energy minimization while satisfying the system reliability constraint. To facilitate the comparison, we assume that the mapping for MVFS is fixed.
- GHO (Greedy-based Hybrid Optimizing) considers also energy, reliability and security factors in one framework. Similar to the SASES algorithm [3], we design GHO as a benefit-cost iteration searching algorithm.
- EO (Energy Optimization) minimizes the energy consumption using DVFS without imposing reliability and security constraints in our framework.
- RSO (Reliability and Security constrained Optimization) does not optimize the energy, but optimizes the security and addresses the reliability constraint in our framework.

For GHO, we set initially the security level of each message as the lowest one, and operating mode of each task as the highest one. In each iteration, we increase gradually the security level by comparing the reduced energy-vulnerability ratio among of all messages and then reduce the operating mode of tasks according to the reduced energy-reliability ratio without violating the deadline and system vulnerability goal. EO and RSO are all using the same TS as presented in Section 6. The difference is in the types of moves that they perform and the cost function used to guide the search. Thus, EO uses the energy *En* as the cost function (ignoring the security and reliability degree terms) and uses only moves to change the operating mode. RSO only considers SecMoves. For EO and MVFS we set the security level L_i of each message to be 1, and then increase it gradually for all critical messages, to meet their individual vulnerability bound VB_i. We used the execution overheads of the RC6 variants measured on a real embedded system environment, which are shown in Table 2. We used also the DVFS-enabled processing nodes given in Table 3 [13]. For simplicity, we scaled the overheads of RC6 variants for other processors by multiplying the relative frequency ratio (compared with 500 MHz).

Based on the proposed TSHO algorithm, we can see that the most important metrics are energy consumption, system reliability and overall vulnerability. Thus, in the experiment we also need to evaluate these three metrics.

- Energy consumption: In the experiment, we will first test the energy cost of all algorithms. RSO always obtains highest energy cost without considering DVFS to reduce the energy cost. Thus, to make the results easier to follow, we normalize the energy costs of all algorithms to the highest value of RSO.
- Reliability: We use the real reliability value (absolute reliability of the whole application) to denote the reliability performance of all algorithms. Moreover, we also introduce $\eta = (1 - RE)/(1 - RE^0)$ to denote the reliability degradation comparing with the given reliability goal.
- Vulnerability: For easier comparison, we use a "Vulnerability Deficiency (VD)" measure to evaluate the system vulnerability performance, namely, $VD = (OV_B - \sum_{m_i \in \mathbb{M}} w_i V_i)/OV_B$. Thus, a value of $VD \geq 0$ means that the system vulnerability requirement OV_B is satisfied. However, a value of $VD < 0$ means that the system vulnerability bound OV_B is not satisfied.

7.1. Impact of application size

We first evaluated the performance of TSHO on applications of 20, 30, 40, 50 and 60 tasks mapped hardware architecture with 2, 3, 4, 5 and 6 processing nodes, respectively. The task graph of each application is obtained with TGFF [24]. We only use TGFF to generate the application graph, which denotes the task number and the dependency among all tasks. The value of other parameters, e.g., task execution time and message size, are additionally generated by our simulated codes. Therefore, the parameters in TGFF need to be configured for each application graph are *task_cnt* (task number) and *task_degree*. In this group of experiments, we set the task number from 20 to 60. We set $task\_degree = (3, 3)$, which means that each task can have up to 3 incoming and 3 outgoing tasks. The size of each message is chosen randomly between 1 and 5 blocks (Note: 1 block equals 16 Bytes for RC6); the weight of each message is set randomly between 0.1 and 1; the WCET of each task is distributed between 10 and 50 μs, referred to the fastest processing node. We assume that one third of all messages are security-critical messages, and the system vulnerability demand is set to be 2 times of the vulnerability when all messages are assigned with security level $L_i = 3$. The number of critical tasks is assigned as half of the total tasks, and each critical task has K =1 or 2 replicas. The deadlines of these five problem sizes are chosen to be 1700 μs, 2100 μs, 3100 μs, 3300 μs, 3500 μs, respectively. On each problem size, 5 different applications with different task WCET, and message size were generated, and the results reported are averages over these five applications.

The reliability goal RE_B of the application has been set such that the probability of failure is less than 20 times of RE^0, where RE^0 is the reliability of a system running with the maximum operation mode. Thus, we have defined the reliability degradation $\eta = (1 - RE)/(1 - RE^0)$, and we have calculated RE_B such that $\eta \leq 20$.

The obtained results of TSHO together with the other approaches used for comparison are presented in Figs. 4–6. As can be observed from Fig. 4, RSO has the highest energy consumption, while the others are approximately on the same level. This is because RSO does not consider DVFS for saving energy. Mean-

Table 3
Parameters of processing nodes.

Fast node			Medium node			Slow node		
Freq. (MHz)	Volt. (V)	Power (W)	Freq. (MHz)	Volt. (V)	Power (W)	Freq. (MHz)	Volt. (V)	Power (W)
500	1.2	9.2	300	1.4	4.3	133	1.1	1.36
600	1.25	12	350	1.5	5.6	166	1.2	1.9
700	1.3	15.1	400	1.6	7.1	200	1.3	2.58
800	1.35	18.6	450	1.7	8.95	233	1.4	3.4
1000	1.4	25	500	1.8	11.4	266	1.5	4.4
$a = -6.5, b = -0.039$			$a = -8.9, b = -0.038$			$a = -6.5, b = -0.039$		

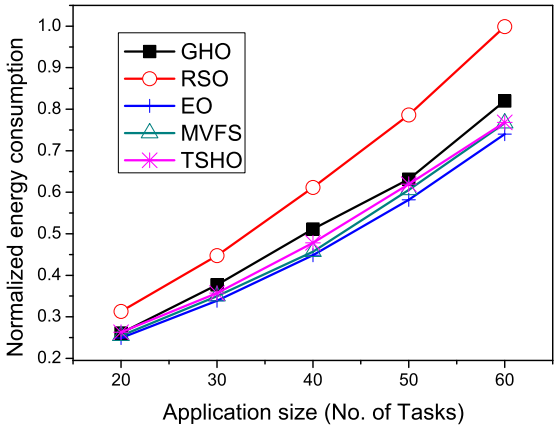


Fig. 4. Impact of application size on energy.

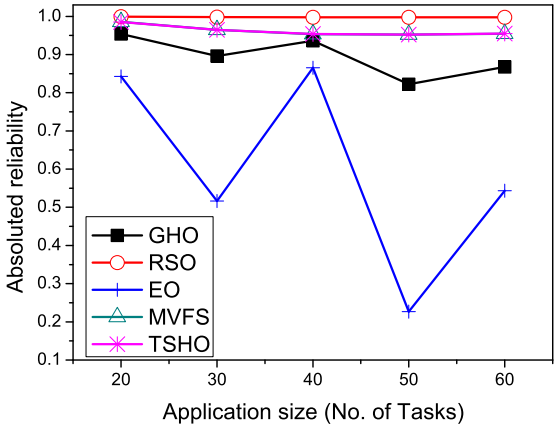


Fig. 5. Impact of application size on reliability.

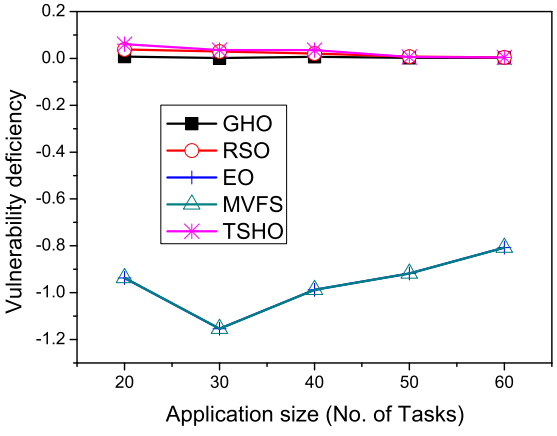


Fig. 6. Impact of application size on vulnerability deficiency.

while, GHO does not make efforts to find the good combination of security level and operating mode, then also results in more energy consumption than TSHO. EO, which does not impose system vulnerability or reliability constraints, obtains the lowest energy consumption. Considering the reliability constraint in the searching procedure, MVFS obtains the energy a little higher than EO on average. Based on the results in Fig. 4, TSHO saves on average 21.3% and 4.4% energy compared with RSO and GHO, respectively. This shows that using TSHO it is possible to reduce the energy consumption while meeting the security, reliability and timing requirements.

Fig. 5 shows that RSO delivers the highest reliability, while EO gives the lowest. The reason is that RSO does not scale the voltage and frequency to sacrifice reliability, and EO ignores totally the reliability factor. GHO also obtains low reliability due to not consider reliability constraint, even though it uses the energy/reliability ratio as one searching metric. MVFS and TSHO have approximately the same reliability results since they are designed to search for solutions within the reliability goal. MVFS and TSHO can get the reliability degradation as 19.7 and 19.9 which are within the predefined 20 times reliability degeneration. The reliability degradation of GHO is 55.7. For EO, it is even $\eta = 211.7$ on average, which means the failure probability of EO is 211.7 times greater compared to RSO, which does not use DVFS.

We have also compared the optimization approaches in terms of security. The results are presented in Fig. 6. We can see from the figure that RSO, GHO and TSHO meet the security requirements (The values of vulnerability deficiency are above zero). RSO does this at the expense of using the largest energy consumption among all the algorithms. However, MVFS and EO, which do not optimize the system vulnerability, are missing the system vulnerability goal, though they are designed to satisfy the individual vulnerability constraint of each critical message.

7.2. Impact of system vulnerability constraint

In this set of experiments, we are interested to see the impact of system vulnerability constraint on one application. In this group of experiments, we configure the task number as 30 and $task\_degree = (3, 3)$ in TGFF generation. The application has 30 tasks that are mapped to 3 processing nodes, i.e., one slow, one medium and one fast processor. In this simulation, we set also a reliability goal which has the reliability degradation $\eta \leq 20$ and $RE_B = 1 - 20 \times (1 - RE^o)$. The number of security-critical messages is set to 10. The application deadline is assigned to 2000 µs. We assume 14 critical tasks, each of which has $K = 1$ replica. For a set of critical messages, the Maximal Overall Vulnerability (MOV) can be calculated when each message are assigned by the lowest security level. In this section we refer the system vulnerability constraint as β, so the system vulnerability can be obtained by $OV_B = \beta \cdot MOV$. We increase the β from 0.2 to 0.7 with a step of 0.1. The other parameters of messages and tasks are set as in the previous set of

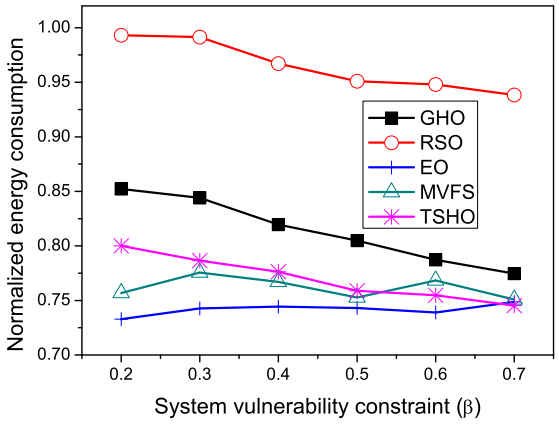


Fig. 7. System vulnerability constraint VS energy.

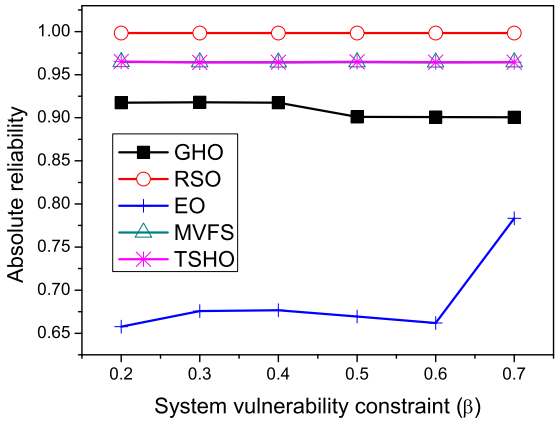


Fig. 8. System vulnerability constraint VS reliability.

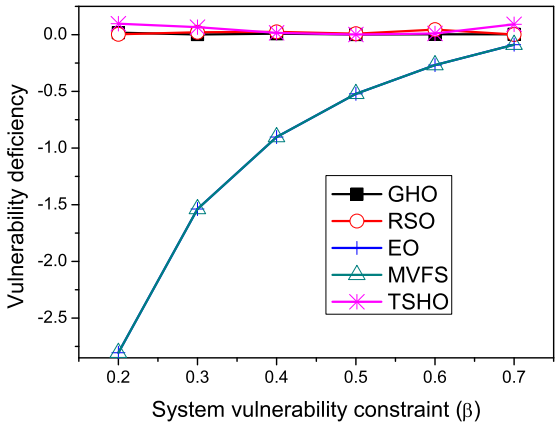


Fig. 9. System vulnerability constraint VS vulnerability deficiency.

experiments. Figs. 7–9 show the system vulnerability impacts to the energy, reliability and vulnerability deficiency, respectively.

As can be seen from Fig. 7, the average energy of RSO is the highest among the four approaches, which is 42,886 μJ. GHO has the second highest energy consumption. Energy consumptions of the other three ones are very close to each other. An interesting decreasing trend with system vulnerability increment can be observed for RSO, GHO and TSHO. This is because a higher system vulnerability happens when there is reduced security protection for messages, which also means more slack for scaling down the CPU frequency (and voltage), that leads to more energy saving. EO and MVFS set the security level only to guarantee the individual vulnerability demand of each critical message, then their energy consumption did not decrease by the increase of system vulnerability constraint. On average, the proposed TSHO saves 20.2% and 5.3% energy compared with RSO and GHO, while achieves 3.8%, and 1.1% more energy than EO and MVFS, respectively.

The reliability results are shown in Fig. 8. RSO has a constant reliability of 0.9982 as no DVFS is deployed. MVFS and TSHO obtain relative low reliability but still manage to meet the reliability goal, having the reliability degeneration as 19.9 and 19.8 respectively. EO has the lowest reliability and the value is averagely 0.687, which means 175.0 times reliability degeneration and seriously violate the reliability goal (20 times degradation). GHO, which obtains the reliability degeneration as 50.8, also misses seriously the reliability goal.

Fig. 9 depicts the security results of the solutions obtained from these five studied approaches. We obtain that RSO, GHO and TSHO have the vulnerability deficiency values as 0.020, 0.004 and 0.028 respectively, meaning all of them satisfy the system vulnerability bound. RSO does this at the expense of using the largest energy consumption among all the algorithms. GHO and TSHO improve the vulnerability by considering the OV_B as a constraint. EO and MVFS violate the system vulnerability goal because they ignore the system vulnerability constraint. The system vulnerability of each of them is 305.8, resulting in the vulnerability deficiency values from -2.81 to -0.087.

7.3. Impact of deadline constraint

In this set of experiments, we are interested to see the deadline impact of one fixed application with given task mapping. Generally, the power of processor includes both dynamic power and leakage power [25]. In this section, besides the dynamic power of processing nodes, we also consider to test the impact of leakage power . The power of node N_j is extend to $pw^{N_j} = pw^{N_j}_{dyn} + Pow^{N_j}_{leak}$. It can be easily extended to support other kind of static power. In this set of experiments, we set the leakage power as 1.2 W, 0.8 W, 0.4 W for fast, medium and slow node, respectively. The application generated by TGFF has 30 tasks that are mapped to 3 processing nodes, i.e., one slow, one medium and one fast processor. In this simulation, we set also a reliability goal which has the reliability degradation $\eta \leq 20$ and $RE_B = 1 - 20 \times (1 - RE^o)$. The number of security-critical messages is set to 10. The system vulnerability is assigned to 200. We assume 14 critical tasks, each of which has $K = 1$ replica. We increase the deadline form 1750 μs to 1950 μs. The other parameters of messages and tasks are obtained with the same criteria as the previous set of experiments. Figs. 10–12 demonstrate the deadline impacts to the energy, reliability and vulnerability deficiency, respectively.

As can be seen from Fig. 10, the average energy of RSO is the highest among the four approaches, which is 34,422 μJ specifically. Energy consumptions of the other four ones are very close to each other. An interesting decreasing trend with deadline increment can be observed for EO, MVFS, GHO and TSHO. This is because longer deadlines leave more slacks for scaling down the CPU frequency (and voltage), that leads to more energy saving. EO consumes the least energy, while the energy expenditures of GHO, MVFS and TSHO are between them. On average, the proposed TSHO saves 19.3% and 5.6% energy compared with RSO and GHO, while achieves 2.2%, and 0.9% more energy than EO and MVFS, respectively. Comparing with previous two sets of experiments, the average energy saving in this section is relatively lower. This is because the leakage power is incorporated to calculate the energy consumption, which results in relative higher energy cost for all algorithms. However, the superiority of TSHO to other algorithms is still obvious.

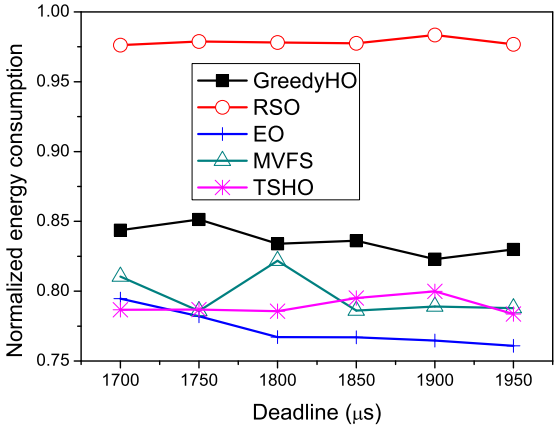


Fig. 10. Impact of deadline on energy.

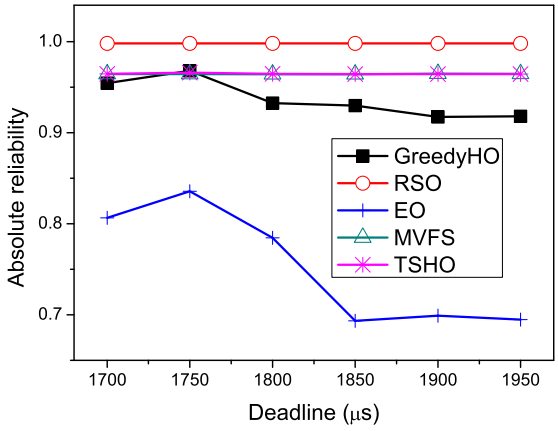


Fig. 11. Impact of deadline on reliability.

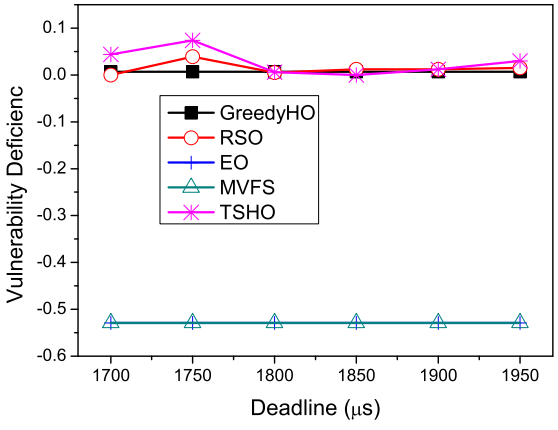


Fig. 12. Impact of deadline on vulnerability deficiency.

The reliability results are shown in Fig. 11. RSO has a constant reliability of 0.9982 as no DVFS is deployed. MVFS and TSHO obtain relative low reliability but still manage to meet the reliability goal, having the reliability degeneration as 19.9 and 19.7 respectively. EO has the lowest reliability and the value is averagely 0.752, which means 138.7 times reliability degeneration and seriously violate the reliability goal (20 times degradation). GHO, which obtains the reliability degeneration as 35.4, also miss seriously the reliability goal.

Fig. 12 depicts the security results of the solutions obtained from these five studied approaches. We obtain that RSO, GHO and TSHO also meet the system vulnerability bound. RSO does this at the expense of using the largest energy consumption among all the algorithms. GHO has the nearly unchanging vulnerability value, which is very close to the vulnerability bound OV_B. EO and MVFS violates the system vulnerability goal due to ignore the system vulnerability constraint as a searching factor. Both EO and MVFS are designed to satisfy the individual vulnerability constraint of each message, thus the system vulnerability deficiency of them are the same and fixed under different deadlines.

According to above three groups of experiments, the conclusions is that: EO and MVFS obtain relatively low security qualities; RSO is obviously not competent for energy-critical SSCSs; for GHO, the energy consumption is still more than TSHO although it satisfies the system vulnerability constraint. The proposed TSHO can achieve significant energy savings while satisfying system vulnerability, deadline and reliability demands, which means that TSHO is the most suitable approach among of them for designing distributed real-time SSCSs.

Table 4
Runtime comparison (ms).

Task no.	TSHO	EO	MVFS	GHO	RSO
20	4422	134	97	145	13
30	41,241	525	477	816	37
40	117,312	1803	1464	1920	110
50	322,926	6195	4396	6653	285
60	878,047	15,212	9422	14,224	603

7.4. Discussion about the efficiency of algorithms

In this section, we discuss about the complexity of these five candidates. TSHO is the Tabu search based hybrid optimization algorithm, which is a globally neighbourhood searching based meta-heuristic. We considered reliability, vulnerability and deadline constraints within TSHO to search the best solution. Thus, TSHO has the highest complexity (both time and memory complexity). Although RSO, MVFS and EO are also Tabu search based methods, they have lower complexity comparing with TSHO. EO focuses on energy optimization with only deadline constraint, which tries to explore the operating mode of all tasks to generate solutions. MVFS has relative lower complexity than EO, because MVFS aims for energy optimization by considering deadline and reliability constraints. Considering reliability constraint in MVFS means that we can utilize reliability violated solutions to diversify the searching space, which will enhance the searching efficiency. RSO has the lowest complexity due to it only considers reliability and deadline constraints to optimize the security performance. We only considered to explore the security assignments to generate highly secure solution in RSO. As a benefit-cost driven greedy algorithm, GHO incorporates all constraints to iteratively search the final solution, whose complexity is relative high.

We also test the runtime of these algorithms by simulation. Based on the same experimental configurations as Section 7.1, we obtain the runtime results of all algorithms for different application size (Task no. changes from 20 to 60), which is summarized in Table 4. We can have following observations. The runtime increases with the increase of application size for all algorithms. This is because larger size will result in higher overhead to search the security and frequency assignments for applications. TSHO has the largest time overhead among these five candidates. RSO takes the lowest runtime to generate the solutions. GHO has higher overhead than RSO. MVFS and EO take higher time overhead than RSO. GHO even takes higher overhead than MVFS and EO. The results of Table 4 also evaluate the former analysis of complexity for all algorithms. Note that, although the runtime of TSHO is much higher than all other candidates, it can also be used for the

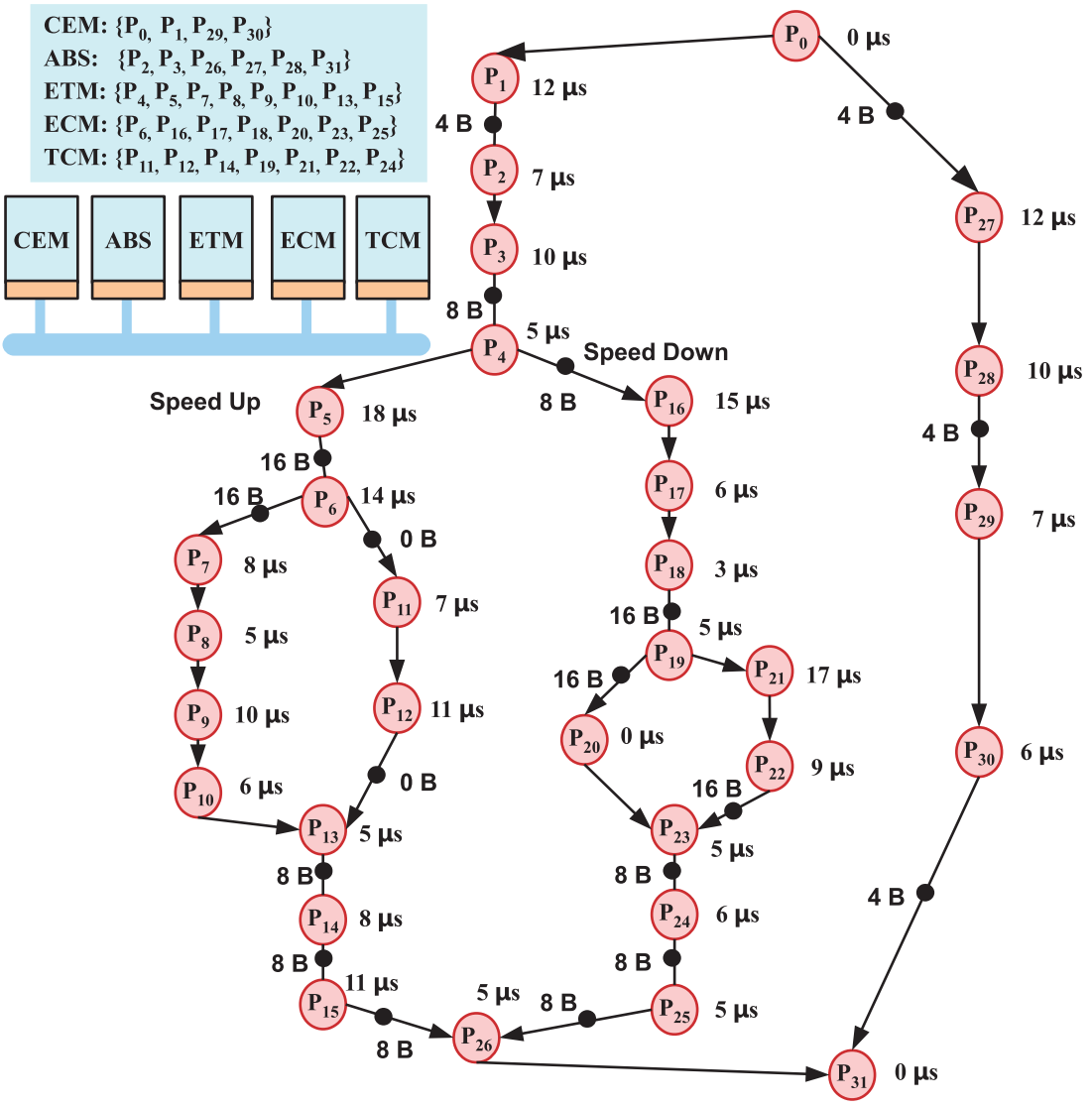


Fig. 13. Vehicle cruiser controller application.

design optimization of real-time applications in SSCSs. The reason is that most design problems are static optimization problems, which can be addressed using high-speed computer before the real implementation.

7.5. A real-life application simulation

In this paper, we also conducted a real-life application, i.e., a vehicle cruiser controller (VCC for short), to evaluate the efficiency of the proposed algorithm. Modern automotive vehicles are emerging to be electricity powered, highly connected by vehicle to vehicle (V2V) and vehicle to infrastructure (V2I), leading to serious challenges on energy efficiency and security protection, besides the typical safety requirements. We consider the VCC system derived from a specification provided by the industry [26]. The main purpose of VCC is to maintain a constant speed for the vehicle and also maintain a safe following distance from the preceding vehicle. VCC can also change automatically the maximum speed depending on the speed-limit regulations and helps the driver with the braking procedure in extreme situations.

The application graph that models the VCC has 32 tasks, and is described in Fig. 13. The VCC was mapped on an architecture composed of five nodes: antilock breaking system (ABS), and transmission control module (TCM), engine control module (ECM), electronic throttle module (ETM), and central electronic module (CEM). In this experiment, we assume all these five nodes have the characteristic of slow node in Table 4. The WCET of each task and the size of each message are associated beside them accordingly. Given the processing nodes, the frequency and the execution of tasks, we also use the reliability model in Section 4 to calculate the system reliability. For the protection of messages, we also consider RC6 variants (in Table 2) to enhance the confidential protection. We have considered the reliability degradation bound as $\eta = 20$, system vulnerability bound as 350.9 (1.2 times of the system vulnerability when all messages are assigned with level 3) and the period (deadline) of VCC as 1000 µs. In VCC benchmark, we have 18 messages that need to be delivered among the common bus with bandwidth 1 MB/S. The weight of each message is assigned randomly within [0, 1]. We set P_0, P_2, P_5, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}, P_{16}, P_{17}, P_{19}, P_{21}, P_{24}, P_{27}, P_{30}, P_{31} as critical tasks, each of which is provided with 1 replica. The VCC application, consisting of 32 tasks to maintain a constant speed, is repeated for every 1000 µs. Obviously, it is a static optimization problem to find out the best security and DVFS assignments for messages and tasks. Thus we can use our TSHO to address the design problem of VCC application.

Table 5
Results for ACC.

	GHO		RSO		EO		MVFS		TSHO	
Metric	value	met?	value	met?	value	met?	value	met?	value	met?
(RE, η)	(0.9819, 61.7)	×	(0.9997, 1)	√	(0.9558, 150.8)	×	(0.9942, 19.8)	√	(0.9943, 19.4)	√
OV	349.4	√	348.3	√	676.4	×	676.4	×	346.3	√
En	12081 μJ		14598 μJ		10897 μJ		10984 μJ		11277 μJ	

The experimental results generated by TSHO and the other four heuristics for one application period are summarized in Table 5. In this experiment, we directly give the values of energy consumption, reliability and overall vulnerability for all algorithms. Note that "met?" means whether the corresponding constraint are satisfied. TSHO produced a schedulable security- and safety-critical implementation with energy as 11,277 μJ, system vulnerability as 346.3. Compared with GHO and RSO, TSHO can save energy by 6.7% and 22.7% respectively. Although EO and MVFS obtain less energy consumption (saving 3.4% and 2.6% comparing with TSHO respectively), both of them cannot provide ideal security protection, which violate seriously the system vulnerability constraint ($OV_B = 350.9$). TSHO is the only approach which can guarantee all the reliability and security requirements while providing low energy expenditure.

8. Conclusion and future work

Security and safety are two emerging requirements for dependable real-time embedded systems with constrained resources. This paper proposed a unified framework for tackling energy, security, reliability and timing requirements for SSCS systems. For communication security, we apply cryptographic algorithms on message transmissions to ensure confidentiality, and introduce a vulnerability based metric to quantify the security strength of communications for distributed systems. Fault-tolerance against transient faults is achieved by task replication. Besides meeting these constraints, our design objective is to minimize the overall energy consumption using DVFS. However, DVFS influences the reliability of the application which is also modelled and studied in this work.

In this paper, our goal was to find the best system designs for security- and safety-critical embedded systems. Our main contributions are: a vulnerability based method is proposed to quantify the security performance of communications on distributed systems; a unified design problem for SSCSs is addressed, which simultaneously considers security, reliability, energy, and timing requirements; an evaluation of the proposed approach on several synthetic benchmarks and one real-life case study (a vehicle cruise controller application). Due to the complexity of solving the problem, we proposed a Tabu Search-based optimization approach TSHO for finding the solution which minimizes energy under the imposed constraints. Experimental results have demonstrated the efficiency of the proposed approach.

For the future work, we are interested in considering simultaneously hardware acceleration of security protection and task replicas into our framework. Although this might result in the increase of additional hardware overhead like FPGA unit, the design trade-off can be well incorporated in the framework. In this paper, we targeted at the design of statically scheduled distributed real-time applications, which are periodically repeated to complete the same mission, e.g., the VCC application. The solution can be generated off-line (statically), and then be deployed to run the application. The method we present here is not suitable for dynamic systems. Therefore, a significant work is to find some efficient methods to address the problem of dynamic real-time applications. Additionally, we would like to extend our framework to support multi-objective optimizations for safety and security embedded applications. Finally, considering security protections to resist Side Channel Attacks like fault injection attack in the system-level will also be our future work.

Acknowledgement

This work was partly supported by the National Natural Science Foundation of China under Grant No. 61003032, the Research Fund of National Key Laboratory of Computer Architecture under Grant No. CARCH201501 the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing under Grant No. 2016A09.

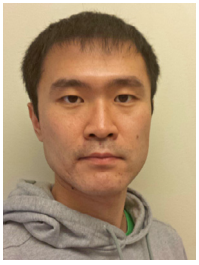
References

[1] C. Constantinescu, Trends and challenges in VLSI circuit reliability, IEEE Micro 23 (4) (2003) 14–19.
[2] F. Sagstetter, M. Lukasiewycz, S. Steinhorst, M. Wolf, A. Bouard, W.R. Harris, S. Jha, T. Peyrin, A. Poschmann, S. Chakraborty, Security challenges in automotive hardware/software architecture design, in: Proceedings of DATE, 2013, 2013, pp. 458–463.
[3] T. Xie, X. Qin, Improving security for periodic tasks in embedded systems through scheduling, ACM Trans. Embed. Comput. Syst. 6 (3) (2007) 20:1–20:20.
[4] M. Lin, L. Xu, L.T. Yang, X. Qin, N. Zheng, Z. Wu, M. Qiu, Static security optimization for real-time systems, IEEE Trans. Ind. Inf. 5 (1) (2009) 22–37.
[5] T. Hoppe, S. Kiltz, J. Dittmann, Security threats to automotive CAN networks practical examples and selected short-term countermeasures, Reliab. Eng. Syst. Saf. 96 (1) (2011) 11–25.
[6] K. Jiang, P. Eles, Z. Peng, Optimization of message encryption for distributed embedded systems with real-time constraints, in: 14th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2011, 2011, pp. 243–248.
[7] K. Jiang, P. Eles, Z. Peng, Co-design techniques for distributed real-time embedded systems with communication security constraints, in: Proceedings of the Conference on Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, 2012, pp. 947–952.
[8] X. Zhang, J. Zhan, W. Jiang, Y. Ma, K. Jiang, Design optimization of security-sensitive mixed-criticality real-time embedded systems, in: the 1th workshop of Real-time Mixed Criticality Systems (ReTiMiCS), 2013, 2013, pp. 12–17.
[9] Y. Wang, H. Liu, D. Liu, Z. Qin, Z. Shao, E.H.-M. Sha, Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip, ACM Trans. Des. Autom. Electron. Syst. 16 (2) (2011) 14:1–14:32.
[10] E. Le Sueur, G. Heiser, Dynamic voltage and frequency scaling: the laws of diminishing returns, in: Proceedings of the International Conference on Power Aware Computing and Systems, 2010, 2010, pp. 1–8.
[11] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: IEEE/ACM International Conference on Computer Aided Design, 2004, 2004, pp. 35–40.
[12] B. Zhao, H. Aydin, D. Zhu, Generalized reliability-oriented energy management for real-time embedded applications, in: 48th ACM/IEEE Design Automation Conference, 2011, 2011, pp. 381–386.
[13] J. Gan, F. Gruian, P. Pop, J. Madsen, Energy/reliability trade-offs in fault-tolerant event-triggered distributed embedded systems, in: Proceedings of ASPDAC, 2011, 2011, pp. 731–736.
[14] V. Izosimov, P. Pop, P. Eles, Z. Peng, Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems, in: Proceedings of DATE, 2005, 2005, pp. 864–869.
[15] E. Bini, G. Buttazzo, G. Lipari, Minimizing cpu energy in real-time systems with discrete speed management, ACM Trans. Embedded Comput. Syst. 8 (4) (2009) 31:1–23.
[16] Make Accurate Power Measurements with NI Tools, 2008, http://zone.ni.com/devzone/cda/tut/p/id/7077.
[17] W. Jiang, Z. Guo, Y. Ma, N. Sang, Measurement-based research on cryptographic algorithms for embedded real-time systems, J. Syst. Archit. 59 (2013) 1394–1404.
[18] R. Chandramouli, S. Bapatla, K.P. Subbalakshmi, Battery power-aware encryption, ACM Trans. Inf. Syst. Secur. 9 (2) (2006) 162–180.

[19] S. Contini, R.L. Rivest, M.J.B. Robshaw, Y.L. Yin, The security of the rc6 block cipher(1998).
[20] L.R. Knudsen, W. Meier, Correlations in rc6 with a reduced number of rounds, in: Fast Software Encryption, 2001, 2001, pp. 94–108.
[21] P. Pop, K.H. Poulsen, V. Izosimov, P. Eles, Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems, in: IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis, 2007, 2007, pp. 233–238.
[22] D. Tămas-Selicean, P. Pop, Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures, in: Proceedings of IEEE 32nd Real-Time Systems Symposium (RTSS), 2011, 2011, pp. 24–33.
[23] P. Eles, Z. Peng, P. Pop, A. Doboli, Scheduling with Bus Access Optimization for Distributed Embedded Systems, IEEE Trans. Very Large Scale Integr. Syst. 8 (5) (2000) 472–491.
[24] R.P. Dick, D.L. Rhodes, W. Wolf, Tgff: task graphs for free, in: Proceedings of the 6th International Workshop on Hardware/Software Codesign, 1998, 1998, pp. 97–101.
[25] X. Pan, W. Jiang, K. Jiang, L. Wen, K. Zhou, Energy optimization of stochastic applications with statistical guarantees of deadline and reliability, in: Proceedings of ASPDAC, 2016, 2016, pp. 12–17.
[26] P. Pop, P. Eles, Z. Peng, Analysis and Synthesis of Distributed Real-Time Embedded Systems, Kluwer Academic Publishers, 2004.

Wei Jiang received the B.S. degree, the M.S. degree and the Ph.D. degree from the University of Electronic Science and Technology of China, Chengdu. He is currently an Associate Professor with the Computer Science and Engineering Department, University of Electronic Science and Technology of China, China. His research interests include real-time system, security/energy aware design, embedded system design, advanced computer architecture and reconfigurable computing. He is a member of ACM and IEEE.

Paul Pop received the Ph.D. degree from Linkoping University, Sweden. He is currently a Professor with the Department of Compute, Technical University of Denmark. His research interests include safety-critical embedded systems, real-time systems, system-level design methodology.



Ke Jiang is associated with AF-Technology AB, Sweden. He received his Ph.D. degree from the Linkoping University, Sweden, in 2016, the M.S degree from Uppsala University, Sweden, in 2009, and the B.S. degree from Hunan University, China, in 2006. His current research interests include design and optimization of secure systems, real-time and low-power embedded systems, and system co-design.