A FRAMEWORK FOR SECURITY AND WORKLOAD GRADUAL ADAPTATION

Antonio Vincenzo Taddeo, Luis Germán García Morales and Alberto Ferrante
ALaRI, Faculty of Informatics, University of Lugano, Lugano, Switzerland

Keywords: Gradual adaptation, Framework, Performances, Workload, Security, Adaptation, Self-adaptivity.

Abstract: Providing a balanced trade-off among performances, security, and energy consumption is a key challenge in embedded systems. Traditional security solutions assume a well-known and static operating environment, thus leading to a static system configuration that cannot be tailed to the system conditions. Wireless sensor networks are a good example of typical embedded systems.
In this work we propose a framework that reduces energy consumption in nodes of wireless sensor networks. The framework allows the system to self-modify its security and workload settings. Adaptations are performed by moving to adjacent configuration and, thus, this mechanism is named *gradual adaptation*. In this paper we discuss the policies that can be used to control the adaptations and we present the results obtained when implementing a case study on Sun SPOT nodes. The results show that the use of the framework increases the energy efficiency of the network nodes. Furthermore, they show the effects of the different policies on the behavior of nodes.

1 INTRODUCTION

Wireless Sensor Networks (WSN) are composed of a large number of nodes that are usually able to perform some measures through suitable sensors, process the gathered data, and send information to other nodes of the network. A number of more powerful devices are also usually in the network with the goal of collecting the data produced by the other nodes. Nodes are placed in the environment that they have to sense and, therefore, they are resource-constrained being limited in terms of area, memory, computation capabilities, and power. Power consumption is always among the most important constraints for WSN nodes (Mura, 2007); power sources can, in some cases, be recharged by means of local power generators (e.g., small solar panels). Though, the batteries might require to last, even for long times, without being recharged.

An important factor that influences design and performance of WSNs is communication security. In particular, security services such as authentication, confidentiality, and availability are critical for sensor networks operating in hostile environments and, at the same time, handling sensitive data. Designing a secure system in these conditions is challenging (Ravi et al., 2004): traditional security solutions are designed by using ad-hoc approaches which offer specific protection against certain attacks (e.g., countermeasures against denial of service attacks). However, they rely on the assumption that the operative environment is well-known and quite static as well. Moreover, some of these technologies have not been specifically developed for embedded systems; in many cases, their adoption in the pervasive world would be impossible due, for example, to high hardware resources requirements (Ferrante et al., 2005).

Typically, when designing secure systems the worst case scenario is considered: the system has to guarantee adequate protection against the strongest possible security attacks. By following this philosophy, security in WSN is typically based on an “on-off” approach: either security is totally ignored or it is enabled with the strongest algorithms available, with a corresponding high consumption of resources. This is generally in contrast with the requirements of a resource-constrained devices such as mobility, flexibility, real-time configuration, open and dynamic operative environment (Keeratiwintakorn and Krishnamurthy, 2006; Grossschädl et al., 2007).

The problem of optimizing resources used for security, yet providing an adequate level of protection, is a hot topic at the moment (Ravi et al., 2004). In particular, the trade-off between energy and performance requirements of security algorithms is of utmost relevance for embedded systems (Chandramouli et al., 2006). As discussed above, each adopted security solution should be a good compromise among factors that are conflicting in nature such as, for example, power consumption and performances. This optimization is a complex task, especially when performed run-time (Chigan et al., 2005; Lighfoot et al., 2007).

In this paper we concentrate on systems that are able to change their configurations at run-time. In particular, we propose a run-time mechanism to deal with the optimization of security and of system workload in accordance both with application security requirements and with system dynamic energy constraints. Our work is based on the "Adequate Protection Principle" (Pfleeger and Pfleeger, 2006) which states that security should be adequately applied to a given context. We implemented such principle by adopting a *gradual adaptation* of application security and of system workload: security is adapted by moving between adjacent configurations that are compatible with application requirements. System workload can be reduced by modifying task periodicity or by suspending some of them.

In WSNs composed of nodes that can be recharged (e.g., by using local generators such as solar cells), our solution increases WSN nodes lifetime in between recharges. Different strategies are used to favor either security or system workload. The strategy to be used and the way in which it will be applied depends on specific node energy conditions and applicative scenario. The solution proposed in this paper has been implemented on Sun SPOT nodes (Microsystems, 2008).

In Section 2 we discuss a number of relevant related works; in Section 3 we introduce our framework and in Section 4 we describe a number of adaptation policies that we propose; in 5 we introduce the energy models used by the framework. In Section 6 we discuss a case study and we show the results obtained when applying the framework on a sensor network based on Sun SPOT nodes.

2 RELATED WORK

The principle of *gradual adaptation* described in this paper can recall the graceful degradation techniques used in fault tolerant systems; in this kind of systems, in fact, performance may be degraded to keep the system operational even in presence of faults (Herlihy and Wing, 1991; Li et al., 2006). Typically this approach does not address the optimization of security.

The challenge of selecting the best set of cryptographic algorithms that optimizes the trade-off between resource requirements and security provided has been tackled in many works. Techniques to minimize the energy consumed by secure wireless sessions have been proposed in (Karri and Mishra, 2002). The authors investigated the selection of encryption algorithms and of key exchange protocols. However, they did not provide explicitly a run-time self-adaptation mechanism; instead, they shown techniques to minimize energy consumption by matching block sizes of message compression algorithms with data cache sizes. In (Chandramouli et al., 2006) a battery power optimizer for wireless networks has been presented. Authors have performed experiments to model the relationship between power consumption and security of cryptographic algorithms. Such information have been used to formulate a knapsack problem and to find the optimal level of vulnerability by changing the number of rounds in cryptographic algorithms. In contrast with our work, neither application requirements nor the possibility of managing changes in cryptographic algorithms were considered. In (Chigan et al., 2005) the authors describe a self-adaptive security framework at protocol level. The mechanism proposed provides the ability to select the optimal set of security protocols with the best security/performance ratio depending on the malicious level of a node neighbors. However, their adaptation is not triggered by an energy consumption constraint, rather on intrusion detection and peer trust evaluation. On the other hand, their systematic approach to security evaluation can be replicated in our framework in order to better define each security level.

Adaptation of cryptographic primitives is instead presented in (El-Hennawy et al., 2004), where the basic idea is to change the AES cryptography key length according to the confidentiality level required by the user. The matching between confidentiality level and key length is done statically and is not changeable at run-time. Moreover, they mainly lead to improve the overall system performances in terms of encryption rate rather than energy consumption.

In (Taddeo et al., 2010) an approach for gradual adaptation is described; the system considered is a sensor node that may be used in multimedia applications. In this paper we propose a more complete and dynamic framework for gradual adaptation of security along with a set of policies for the system. This includes, as explained in the following of this paper: adaptation of sample period for periodic tasks; dynamic energy budget computation; dynamic monitoring period; adaptation decisions policies specified for each task by means of its requirements.

3 THE FRAMEWORK

In this section we describe our framework for gradual adaptation. This framework has been designed for extending battery lifetime in WSN nodes, yet respecting the constraints fixed by the application designers.

Our framework contains a number of software components that monitor the execution of the tasks in the nodes, decides the possible adaptations on system configuration, and applies them. The current node conditions and the *policies* – directives and constraints previously defined by system and application designers – are used to decide when an adaptation is necessary and how it should be performed. Adaptations consist in changing task parameters such as level of security, execution period, and execution state. The purpose of the framework is to set the best security configuration, and, at the same time, extend the battery life. Figure 1 shows a general scheme for the framework with its main blocks. The framework is composed of two main blocks: the centralized controller and the task library.

3.1 Centralized Controller

The centralized controller monitors the system behavior periodically and performs suitable downgrades or upgrades on the task parameters with the goal of meeting the battery lifetime defined by system designers. The controller computes the energy available in each monitoring period by considering the energy remaining in the battery and the lifetime constraint. This amount of energy is called *Energy Budget*. When the energy consumption is higher than the energy budget, the controller downgrades the parameters (security, execution period, or execution status) of some tasks to reduce energy consumption. In the same way, when the energy consumption is lower than the energy budget, the controller upgrades the parameters of tasks to bring them closer to their ideal configuration. The parameters to be changed for each task are determined by the controller depending on the policies defined by application designers. These policies are called *Tasks Policies*. The tasks are selected for adaptation based on a node-level policy named *Framework Policy* and on a task energy estimator. The estimator monitors the execution of tasks and it provides the energy figures of each one of them. Policies are discussed in Section 4.

3.2 The Task Library and Resources Interface

The task library and resources interface provides the ability to install applications and to define their policies. Furthermore, this library is used to determine the energy spent by tasks during their execution.

The task library is composed of a collection of methods and properties. The tasks are created by extending one of the objects provided in this library. The designer, while extending these objects, needs to specify the kind of task to be created (periodic or non periodic) and the required policies.

Additionally, the framework provides a resource interface that makes easy the access to the different software and hardware resources (e.g. security algorithms). This interface is composed of a collection of functions named “intermediate functions”. They are in charge of calling the real resources and also collecting the execution behavior of every resource with the intention of estimating the energy spent by each one. This information collected and the resource model are used by the estimator when the monitoring process is being performed.

3.3 Security Services

Security can be provided through suitable services called *security services*. These services can be of different kinds such as, for example, encryption/decryption, authentication, and access control. In this work we only consider encryption algorithms as possible security services. Authentication algorithms could be considered as well without any change in the framework. Other services are not relevant in the application field considered in this work.

We divided the encryption algorithms considered in three groups, depending on their level of security. These algorithms are defined to have a high, medium, or low level of security based on their resistance to known attacks: the higher the number of combinations to be tried for breaking the algorithm, the higher its security. Inside each group a sorting is performed based on the energy consumption measured while encrypting/decrypting each byte for each algorithm.

Each task has a range of security levels defined by the application programmers. At the beginning of the execution, the framework assigns to the task the best security algorithm available and compatible with this range. When a downgrade on security is required for the considered task another security algorithm might be selected. Some degradations are *intra-level* (i.e., a new, less energy hungry, security algorithm is selected, but the algorithm belongs to the same security level), while some others are *inter-level* (i.e., the new security algorithm may belong to a different security level). The same applies for security upgrades.

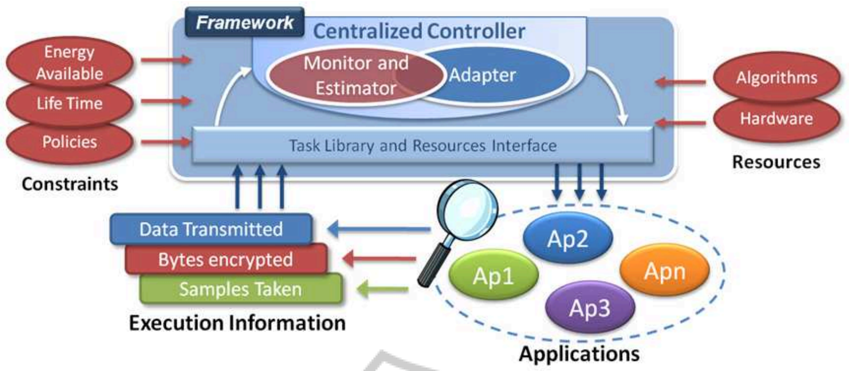


Figure 1: The framework.

4 POLICIES

In the framework different decisions are taken about security and performances adaptations. These decisions are governed by policies that describe what can be done for each task (i.e., if and how its security and performance can be modified) and how the framework should work (i.e., which are the adaptations to be performed, in which order, and with which speed). Policies for the tasks are named *Task Policies*; policies that govern the framework are called *Framework Policies*.

4.1 Task Policy Parameters

The task policies allow designers to specify how security and performance can be changed for each running task. More specifically, the elements that can be altered by the framework are related to the desired level of security, the execution period (for periodic tasks), and the execution status (i.e., the possibility to suspend tasks or not). Therefore, each policy contains five parameters as shown in Table 1.

4.2 Framework Policies

The framework requires proper rules to govern the adaptations and to monitor the execution of tasks and their energy consumption. The rules should provide both information on the tasks that should be considered for degradation and on how to monitor them. The parameters that compose framework policies are listed below:

- **Task Selection.** This parameter tells the framework which are the tasks that should be considered for degradation. We consider four different options:
  - *Most-Energy-Demanding tasks*. Energy consumed by each task is estimated at runtime. The tasks with highest energy demand are considered.
  - *Least-Degraded tasks*. The tasks that are in lowest levels of degradation (i.e., the ones that are closer to their optimal performances) are considered.
  - *Lowest-Priority first*. Degradation is first performed on tasks with the lowest priority level (low); degradations are done on tasks with increasing priority levels if necessary.
  - *Last-Recently-Started tasks*. The tasks that have been started more recently are degraded.
- **Adaptation Speed.** This parameter specifies the percentage or the maximum number of tasks to be adapted in each control period, regardless the adaptation policy considered. This parameter contributes significantly to the speed of adaptation.

The *monitoring period* used by the self-adaptation mechanism has to be defined properly: a too short monitoring interval can lead to excessive energy and computational overhead on the system; a too long monitoring interval can lead, instead, to far from optimal timing on the adaptation of tasks, thus decreasing the effectiveness of the self-adaptation mechanism. A correct trade off between these two parameters should be found to obtain both efficiency and effectiveness of the self-adaptation mechanism. To this end, the framework allows the designer to specify a range of values for the monitoring period. At run-time, the self-adaptation mechanism starts with the minimum monitoring period. If no adaptations occur, the monitoring period is increased in the specified range. When adaptations are necessary, the monitoring period is decreased to allow a more frequent analysis of system conditions. The range to be used for the monitoring period should be determined by the

Table 1: Task policy.

Parameter	Description
Security	Kind of degradation that can be applied to security. The options available are: • no degradation; • intra-level degradations only; • intra and inter level degradations; optionally the range of security levels accepted can be specified.
Period	Range of periods accepted by the task.
Execution status	Sets if the task accepts temporarily suspensions.
Task priority	Sets the priority of the task (High, Medium, and Low) with respect to adaptations.
Adaptation order	Specifies the order in which the adaptations are made. There are five different options available: • *Auto*: the framework has the freedom to decide which parameters to downgrade. • *First security, then the others*: the framework downgrades security first; when it is no longer possible to downgrade security, the period is downgraded; as a last option, if the other parameters are no longer degradable the task is suspended. • *First period, then the others*: this policy is similar to the previous one, but the period is adapted first. • *Suspend only*: whenever an adaptation is required, the task is suspended without considering the security and the period. • *None*: no degradation is performed on the task.

designer at design-time by considering the characteristics of the running tasks and the applicative scenario. For example, the minimum monitoring period could coincide with the minimum period among all the periodic tasks.

5 ENERGY ESTIMATION

In our framework the run-time monitoring of energy consumption is performed through a piece of software named *energy estimator*. The aim of the estimator is twofold: on the one hand, to estimate the energy consumption of each task; on the other hand, to compute the *energy budget* associated with each monitoring interval. These pieces of data are used by the framework to perform the related adaptations.

5.1 Energy Consumption Models

In each node we can consider two different types of energy consumptions: static and dynamic. Static energy is the energy that is necessary to provide the basic node functionalities. This energy cannot be changed for a given node and operating system. Dynamic energy is the one used to execute tasks. Our framework tries to optimize the dynamic energy.

Energy consumption of tasks is estimated through a model that includes the most significant contributions to energy consumption. We considered the following contributions to task energy consumption:

- Radio, $E_{radio}(b)$: the energy required by the radio module to transmit a certain amount of bytes b.
- Sensors, $E_{sensor}(n_s)$: energy consumption required to acquire n_s samples in the observation period.
- Encryption/decryption, $E_{algo}(b)$.

When no task is being executed the node can be switched to one of the low-power modes. In this case the energy consumption becomes $E_{mode}(t)$.

The energy consumption of cryptographic algorithms are estimated by considering the following contributions: the energy spent for initializing the algorithm key, E_{init}; and the energy per byte consumed to processing a block of 16 bytes, E_{block}.

$$E_{alg}(b) = E_{init} + E_{block} * b = \quad (1)$$

$$= E_{init} + (E_{padding} + E_{call} + E_{enc}) * b \quad (2)$$

where b is the number of blocks. Moreover, E_{block} can be further divided into the energy spent for: padding, $E_{padding}$; function calling, E_{call}; encryption, E_{enc}.

All the aforementioned models (excluding E_{mode}) refer to the dynamic energy consumption.

The energy consumed by each task can be estimated as the sum of the different contributions mentioned above. Similarly, the total energy consumption of the system within an observation period can be estimated as the sum of the energy consumption of all tasks running on the system and the energy spent in the current power mode (if enabled).

All of the aforementioned power consumptions are estimated by using linear models (see Table 2for a complete model description). Using a linear equation for modeling the energy consumption of a security algorithm has been proved to be quite accurate (Grossschädl et al., 2007; Keeratiwintakorn and Krishnamurthy, 2006). Instead, linear equations might not be as accurate for other parts of the energy consumption. Such models, in fact, are technology-dependent and have to be characterized according to the particular node technology considered. Our framework relies on an energy consumption model to take decisions on adaptations and the accuracy of the model can influence the efficiency of the adaptation mechanism. Though, different energy consumption models can be easily inserted in the framework without any change in the adaptation logic.

5.2 Energy Budget Computation

For each monitoring interval, the framework computes the total energy consumption based on real consumption. The framework, then, evaluates if the measured energy consumption is compatible with the desired system lifetime. When the energy consumption is evaluated to be too high, the system must gradually reduce it during the successive monitoring intervals. To evaluate if the energy consumption is compatible with the desired battery lifetime, the framework compares it with the reference discharge curve specified by the system designer. An example of a discharge curve is shown in Figure 2. According to this figure, in the monitoring intervals I_1 and I_2, the current energy consumed (continuous line) is lower then the reference one (dashed line). Thus, the system does not need any adaptation. Starting from interval I_3, the current energy consumption becomes higher than the desired one; in this case adaptations are necessary to lower the energy consumption.

In this work, we define the *energy budget* as the amount of energy that can be consumed in a given monitoring interval according to the reference curve and the real energy consumed in previous monitoring intervals. The energy budget is compared with the current overall energy consumption; the result of this comparison drives the adaptations performed by the framework.

If a linear reference discharge curve is considered, the energy budget can be computed as follows. Considering Figure 2, the reference curve is given by:

$$\bar{E}(t) = \frac{minEC - maxEC}{t_{lifetime}} * t + maxEC \qquad (3)$$

Thus, the reference consumption in a given interval [j,j+1] is:

$$\tilde{E}_{[j,j+1]} = \bar{E}(t_j) - \bar{E}(t_{j+1}) \qquad (4)$$

Therefore, the energy budget E_{budget} can be computed as:

$$E_{budget,[j,j+1]} = \tilde{E}_{[j,j+1]} + E_{prev,[j-1,j]} \qquad (5)$$

where $E_{prev,[j-1,j]}$ is the energy budget not consumed in the previous interval. When the energy consumed in the previous interval is higher than the associated energy budget, E_{prev} becomes negative value to take into account this extra energy consumption. Note that, the framework allows the designer to specify his

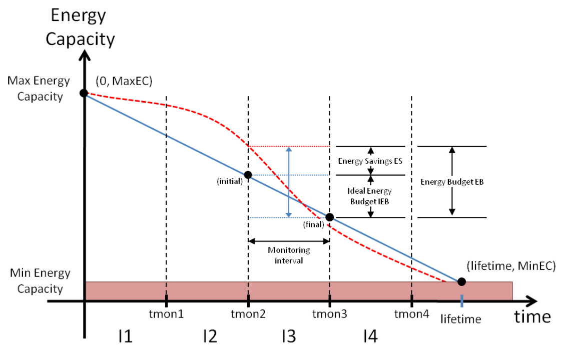


Figure 2: Energy budget computation: parameters and conceptual scheme. Dashed-line is the real energy consumption; continuous-line represents the reference discharge curve.

custom discharge model and corresponding computation of the energy budget keeping the validity of self-adaptation approach here proposed.

6 CASE STUDY

Our framework has been implemented and tested in a real WSN. In this section we describe this implementation and the results obtained by running a number of experiments on this WSN. The goal of the experiments was both to measure the overhead introduced by the framework and to evaluate the effects of different policies on energy consumption and on performances.

6.1 Scenario

The network is based on the Sun SPOT (Sun Small Programmable Object Technology) nodes (Microsystems, 2008). Sun SPOTs are small wireless devices, compliant with the IEEE 802.15.4 standard and running the Squawk Java Virtual Machine (VM) without any underlying OS. The VM acts as both operating system and software layer to provide high portability and fast prototyping capabilities. The Sun SPOTs platform is designed to be a flexible, capable of hosting widely differing application modules. From the hardware point of view, Sun SPOT nodes are equipped with: a 32 bit ARM920T working at 180 MHz; 512K RAM/4M Flash Memory; 2.4GHz IEEE 815.4 radio with integrated antenna; 720 mAh as maximum battery capacity. By default, these devices are equipped with an accelerometer, a temperature, and a light sensor. Sun SPOT nodes may use two different low power modes: *shallow sleep* and *deep sleep* mode.

Concerning security suites, Sun SPOTs support the TLS/SSL protocol with several cryptographic algorithms: AES with 128, 192, and 256 bit keys and RC4 with 40 and 128 bit keys.

The applications in this scenario are classified in three groups according to the different sensors (light, temperature, or acceleration) that they use. Each group contains three tasks, each one performing one of the following operations:

1. Obtaining data from the environment periodically (short period).
2. Encrypting and transmitting the measures previously collected periodically (large period).
3. Detecting possible alarm conditions in the environment (e.g. fire condition). As soon as an alarm is detected, a picture of the environment is taken and transmitted, in an encrypted form, to the other nodes in the network. Since the SunSPOT does not have a camera to take pictures, this functionality is emulated by taking several samples using the respective sensor (temperature).

Different tasks have been implemented as different threads.

6.2 Profiling and Modeling

To determine the parameters of the energy models of Section 5, we profiled the different elements of our nodes. The average values obtained are shown in Table 2. Static energy consumption has been measured to be about 44% of the total energy consumption.

The cryptographic algorithms available in the TLS implementation of SunSPOTs have been also profiled to determine their energy consumptions and the execution times. By using these information we could order the algorithms as explained in Section 3.3. The algorithms have been grouped in three security levels as follows:

- *High*: AES-256, AES-192;
- *Medium*: AES-128, RC4-128;
- *Low*: RC4-40.

6.3 Results

Two different sets of experiments have been performed to evaluate the framework overhead and the effects produced by applying different policy combinations.

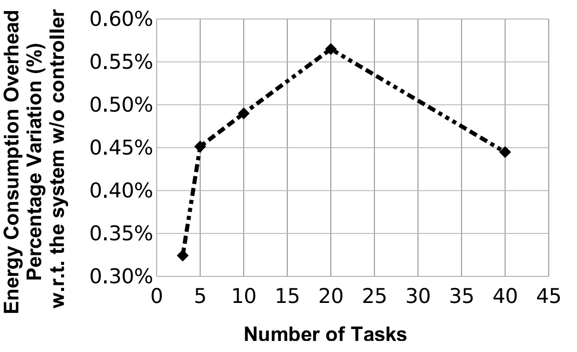
6.3.1 Framework Overhead

The monitoring and adaptation activities performed by the framework introduce an overhead in the system in terms of time and energy. In order to measure this overhead, we measured the energy consumed both when the framework was completely disabled and when the framework was enabled. In both cases we considered tasks programmed to always perform the same actions and to make use of all system resources (sensors, security algorithms, and the radio). In order to study the overhead scalability, measurements have been collected by repeating the experiment with increasing number of tasks.

When the framework was enabled the following additional conditions applied:

- The framework was forced to downgrade all the tasks running over the system in every monitoring interval (worst case scenario assumption).
- The framework was forced to adapt all the available parameters (e.g., security, period and execution state).
- The framework applied fake adaptations over the tasks. In this way both task and framework performance are kept constant along the experiment execution.

In Figure 3 the energy consumption and the execution time overhead are shown. The overhead introduced by enabling the framework increases when



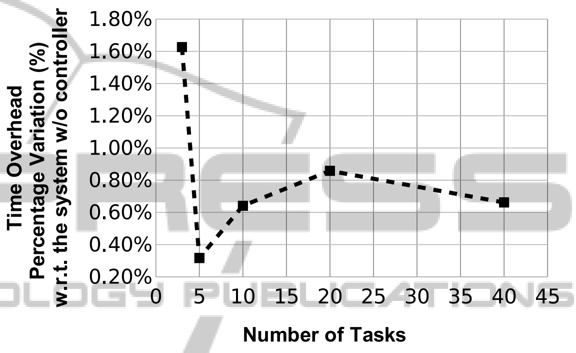


Figure 3: Framework energy consumption (up) and execution time (down) overheads, with respect to the system w/o the framework.

Table 2: Value of parameters for energy consumption models described in Section 5. To model each resource energy consumption a linear model ($y = a + bx$) has been considered.

RESOURCES	a	b
Shallow sleep mode	$E_{sm,on/off} = 0$ [mAh]	$E_{status} = 8.13E-3$ [mAh/s]
Deep sleep mode	$E_{dm,on/off} = 8.22E-3$ [mAh]	$E_{status} = 5.27E-6$ [mAh/s]
Radio	$E_{r,on/off} = 6.40E-5$ [mAh]	$E_{process} = 5.60E-7$ [mAh/bytes]
Temp. sensor	$E_{ts,on/off} = 0$ [mAh]	$E_{sample} = 1.41E-6$ [mAh/sample]
Light sensor	$E_{ls,on/off} = 0$ [mAh]	$E_{sample} = 1.36E-6$ [mAh/sample]
Accel. sensor	$E_{as,on/off} = 0$ [mAh]	$E_{sample} = 4.34E-6$ [mAh/sample]

Table 3: Values of factors in Eq. 2.

Factor	**AES-256**	**AES-192**	**AES-128**	**RC4-128**	**RC4-40**
E_{init} [mAh]	1.39E-5	1.28E-5	1.14E-5	2.25E-5	2.26E-5
$E_{padding}$ [mAh/16B]	9.40E-8	2.77E-7	3.27E-7	1.27E-7	9.40E-8
E_{call} [mAh/16B]	4.55E-7	2.13E-7	3.89E-7	4.29E-7	3.44E-7
$E_{processing}$ [mAh/16B]	9.89E-6	8.65E-6	7.37E-6	1.57E-6	1.59E-6

the number of tasks in the system is increased. In terms of energy the overhead ranges from 0.3% to 0.6%. These two values correspond to 3 and 30 tasks in the system, respectively. In terms of time the overhead ranges from 0.3% to 1.6%. We can conclude that the overhead introduced by the framework is almost negligible in terms of energy and very small in terms of time. The overhead is highly compensated by the added system flexibility.

6.3.2 Effects of Different Policies

The purpose of this set of experiments was to verify the effects of the different combinations of policies on the execution of tasks and on battery life. In these experiments we considered a node equipped with three sensors, each one associated with three tasks. Each of these tasks had different task policies corresponding to different settings for security, period, and execution status. To simulate the most demanding conditions for the framework, we anyway specified policies providing a high degree of flexibility.

We considered 6 different experiments in which all the different policies described in Section 4, except the *Last-Recently-Started* tasks, are combined. The parameters used in these experiments are listed in Table 4. The adaptation speed refers to the percentage of tasks that can be adapted in the set of candidates: *fast* corresponds to 70% of possible candidates; *slow* corresponds to 20%.

For these experiments we fixed a desired system lifetime of 12 hours. The system without our adaptation framework, as well as in cases #4 and #5, was not able to respect this constraint. In all the remaining cases the constraint on battery life is satisfied.

In order to compare the different cases, we considered the impact of each policy in terms of degradation level reached by all the tasks within the observation time frame. Figure 4 shows the percentage of time (computed over the total simulation time) in which each task has been degraded to some degree (e.g., degradations of security, workload, or execution status). Generally, our self-adaptation mechanism keeps tasks in one of their possible degradation levels for a period that spans from 10% to 48% of the total time.

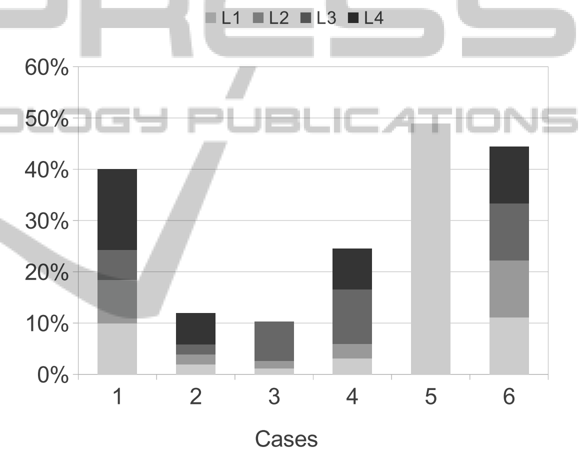


Figure 4: Percentage of time, aggregated by case, all tasks are at a given degradation level w.r.t. total simulation time.

As expected, the cases with a *low* adaptation speed show fewer degradations compared to others. *Most-Energy-Demanding* and *Least-Degraded* criteria show a similar behavior from the number of degradation stand point.

Figures 5, 6 and 7 show details about how the degradation of security has been performed for each task according to different adaptation policies. Values are the percentage of total time in which a task

Table 4: Different combinations of policies used in the simulations.

Case	1	2	3	4	5	6
Adaptation Speed	fast	slow	fast	slow	fast	slow
Criterion	most energy dem.		low priority		less degraded	

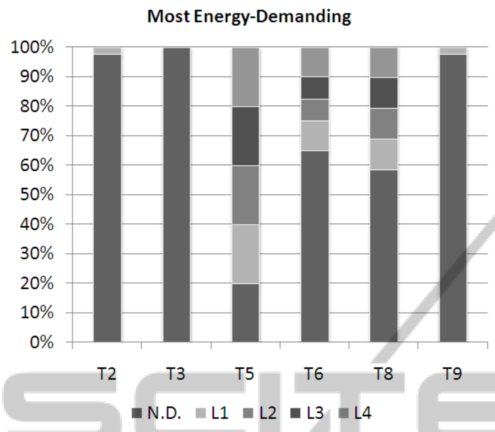


Figure 5: *Most-Energy-Demanding* policy (case #2): percentage of time in every security degradation level per task. Only tasks that support security degradation are shown.

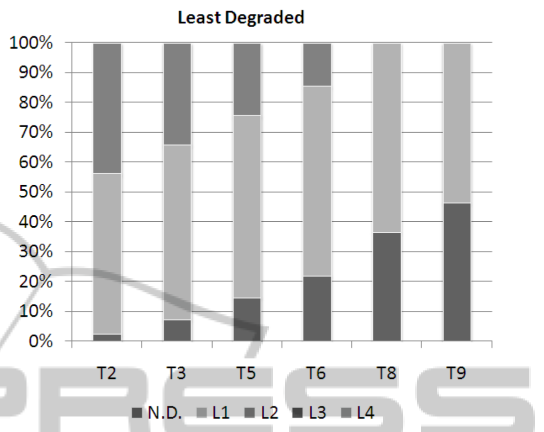


Figure 7: *Least-Degraded* policy (case #6): percentage of time in every security degradation level per task. Only tasks that support security degradation are shown.

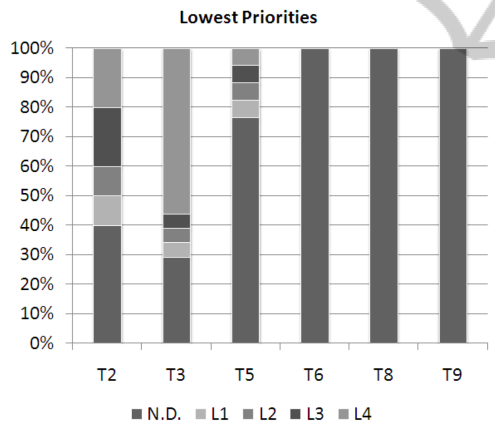


Figure 6: *Lowest-Priority* policy (case #4): percentage of time in every security degradation level per task. Only tasks that support security degradation are shown.

has been in a given degradation level. In general our framework was able to meet the system lifetime constraint even by using optimal or almost optimal settings for the tasks in most of the time. The *Most-Energy-Demanding* criterion provides the best results concerning energy consumption. Though, this is achieved by heavily degrading security of most energy demanding tasks. The *Least-Degraded* criterion provides the best balance in terms of saved energy and degradation level of tasks: in this case the optimization of energy consumption is obtained by distributing a low level of degradation on all tasks.

The aforementioned results also provide some guidelines for the choice of different adaptation policies. The *Most-Energy-Demanding* policy is suitable when energy is the most important constraint. This policy, in fact, tends to limit the energy consumption of the most power-hungry tasks, even by heavily limiting their performances. The *Least-Degraded*, instead, is the policy with the lowest influence on task performances. Therefore, it is suitable when energy consumption should be limited, but performances remain the most important requirement. The *Last-Recently-Started* tasks policy instructs the system to perform adaptations on the tasks started more recently, supposedly the ones that are "additional" (i.e., not the basic ones that had been started at system boot). An adaptation policy based on task priorities (*Lowest-Priority*) gives the programmers flexibility, but it is also complex to design.

7 CONCLUSIONS AND FUTURE WORK

In this paper we discuss a framework for gradual adaptation of security and of system workload. The framework tunes the execution of tasks and their security settings runtime with the goal of optimizing the trade-off among performances, security, and energy consumption. Different policies for controlling the framework along with a model for energy consumption estimation have been also proposed in the paper.

The framework has been tested on a real system along with the policies.

Although the framework introduces a small, but inevitable, energy and time overhead, it also allows the nodes to meet battery lifetime constraints, yet providing an adequate level of security to tasks.

Future work includes the extension of our framework to support smarter parameters for upgrade policies. Learning mechanisms may be introduced to further optimize the gradual adaptation mechanism.

REFERENCES

Chandramouli, R., Bapatla, S., Subbalakshmi, K. P., and Uma, R. N. (2006). Battery power-aware encryption. *ACM Trans. Inf. Syst. Secur.*, 9(2):162–180.

Chigan, C., Li, L., and Ye, Y. (2005). Resource-aware self-adaptive security provisioning in mobile ad hoc networks. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, pages 2118–2124.

El-Hennawy, M. E., Dakroury, Y. H., Kouta, M. M., and El-Gendy, M. M. (2004). An adaptive security/performance encryption system. In *Proc. International Conference on Electrical, Electronic and Computer Engineering ICEEC '04*, pages 245–248.

Ferrante, A., Piuri, V., and Owen, J. (2005). IPSec Hardware Resource Requirements Evaluation. In *NGI 2005*, Rome, Italy. EuroNGI.

Grossschädl, J., Szekely, A., and Tillich, S. (2007). The energy cost of cryptographic key establishment in wireless sensor networks. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 380–382, New York, NY, USA. ACM.

Herlihy, M. and Wing, J. (1991). Specifying graceful degradation. *Parallel and Distributed Systems, IEEE Transactions on*, 2(1):93–104.

Karri, R. and Mishra, P. (2002). Minimizing energy consumption of secure wireless session with qos constraints. In *Proc. IEEE International Conference on Communications ICC 2002*, volume 4, pages 2053–2057.

Keeratiwintakorn, P. and Krishnamurthy, P. (2006). Energy efficient security services for limited wireless devices. In *Proc. 1st International Symposium on Wireless Pervasive Computing*, pages 1–6.

Li, J., Song, Y., and Simonot-Lion, F. (2006). Providing real-time applications with graceful degradation of qos and fault tolerance according to (m, k) -firm model. *Industrial Informatics, IEEE Transactions on*, 2(2):112–119.

Lighfoot, L., Ren, J., and Li, T. (2007). An energy efficient link-layer security protocol for wireless sensor networks. In *Electro/Information Technology, 2007 IEEE International Conference on*, pages 233–238.

Microsystems, S. (2008). Sun Small Programmable Object Technology.

Mura, M. (2007). Ultra-low power optimizations for the ieee 802.15.4 networking protocol. In *proceedings of MASS*.

Pfleeger, C. P. and Pfleeger, S. L. (2006). *Security in Computing, 4th Edition*. Prentice Hall PTR, 4th edition.

Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S. (2004). Security in embedded systems: Design challenges. *Trans. on Embedded Computing Sys.*, 3(3):461–491.

Taddeo, A. V., Micconi, L., and Ferrante, A. (2010). Gradual adaptation of security for sensor networks. In *IEEE WoWMoM 2010: Proceedings of the IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*, Montreal, Canada.