Impact of Operating Systems on Wireless Sensor Networks (Security) Applications and Testbeds

Cíntia B. Margi,
Bruno T. de Oliveira
and Gustavo T. de Sousa
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo
São Paulo, Brazil
Email: {cintia,brunotrevizan,
gustavot}@usp.br

Marcos A. Simplicio Jr,
Paulo S. L. M. Barreto
and Tereza C. M. B. Carvalho
PCS - Escola Politécnica
Universidade de São Paulo
São Paulo, Brazil
Email: {mjunior,pbarreto,
carvalho}@larc.usp.br

Mats Näslund
and Richard Gold
Ericsson Research
SE-16480
Stockholm, Sweden
Email: {mats.naslund,
richard.gold}@ericsson.com

***Abstract*—Wireless Sensor Networks (WSNs) are a valuable technology to support countless applications in different areas. Given the WSN nodes resource constrained characteristics, designing energy-aware applications, communication protocols and security mechanisms are critical. The operating system (OS) running on the WSN node also interferes with the node overall behavior, and its energy consumption. In this paper, we develop a comparison of two different operating systems (Contiki and TinyOS) running on the same hardware platform (Crossbow TelosB). Using a set of tasks, which include sensing, communication and security mechanisms, we evaluate their behavior in terms of energy consumption and execution time.**

I. I

Wireless Sensor Networks (WSNs) are a valuable technology to support the development of new applications in many different areas, such as: environmental and habitat monitoring, surveillance, indoor climate control, structural monitoring, medical diagnostics, disaster management, and so on [1], [2]. They can be defined as a special type of multi-hop ad-hoc networks, and participating nodes are typically battery operated, thus having access to a limited amount of energy. Moreover, these nodes often exhibit additional constrains, such as limited processing, storage and communication capabilities [3].

In several WSN-based applications, the nodes are left unattended for their whole operational lifetime after deployment. Moreover, the sensing and processing tasks the node will execute, as well as the overhead introduced by the node's operating system (OS), must be accounted for in the energy consumption. Hence, understanding the impact of processing and communications tasks on the nodes' energy usage provides the necessary knowledge to choose the appropriate duty cycle. This approach allows the sensor network node to alternate between active, idle and low-power periods, thus saving energy. Indeed, the careful choice of the duty cycles is a common power conservation approach in real deployments [4], [5].

Most WSN deployments do not consider security among their requirements because of the execution time/energy overhead it adds to the system; hence, security tends to be considered simply as an undesirable "extra cost" in such constrained environments. However, when targeting WSNs for health applications or applications that monitor sensitive information, it is important to consider confidentiality; therefore, in these scenarios, the deployment of encryption algorithms is essential. Furthermore, in many situations data integrity and authenticity are also critical, since the existence of invalid data could lead to mistaken actions with severe consequences; since the origin of such invalid information can be either natural (caused by hardware malfunctioning, transmission errors, etc.) or intentional (e.g., generated by fake sensors introduced in the network with malicious intent), the deployment of Message Authentication Codes (MACs) is also made necessary. Finally, since such algorithms depend on the existence of secret keys for their functioning, applications need to deal with the distribution of such keys, which is a especially challenging issue in WSNs [6].

Due to the constraints intrinsic to WSNs, security mechanisms employed must be extremely lightweight. The literature includes some comparative evaluations of traditional hash functions, block and stream ciphers [7], [8], [9], [10] in sensor networks. The main goal of these analyses is to determine the most adequate algorithms for deployment in WSNs, considering both efficiency and security requirements. Similarly, many authors have tried to identify efficient implementations of Elliptic Curve Cryptography (ECC) for such constrained scenarios [11], [12], [13]. The results obtained in these studies usually show that the deployment of general-purpose algorithms leads to the need of choosing performance over security level or vice-versa. For example, after analyzing the efficiency of several traditional block ciphers in WSNs, Law et al. [7] recommends AES [14] and Skipjack [15] for scenarios with, respectively, high and low security requirements. Due to this urge for security in many important scenarios and to the need of sufficiently lightweight solutions, some algorithms specifically designed for WSNs have been proposed and analyzed in the last years, such as TinyTate [11] (for key management), C [16], [17] (block ciphers) and M [18] (a MAC algorithm).

Despite their importance for providing further information about the impact of security mechanisms on WSNs, these studies lack a more detailed comparison with core tasks not related to security, such as sensing and transmitting data. In this work, we focus on the performance evaluation of basic tasks (such as sensing and communication) and the execution of security algorithms on our WSN testbed, as well as the overhead introduced by the Operating System. We consider the same set of experiments on both TinyOS [19] and Contiki [20] operating systems, running on the Crossbow TelosB sensor node. Since the same hardware platform is used, we are able to see how the tasks behave in the different OS, and how that affects the performance of the sensors. Our metrics are the execution time and energy consumption of common tasks (communication and sensing), which are compared with encryption algorithms (namely, C , Skipjack and AES), Message Authentication Codes (M and CMAC [21]), and authenticated-encryption schemes (OCB [22] and L - S [18]). Therefore, our main contribution in this work is the performance analysis of main WSN tasks (i.e., sensing, processing and communication) running in two different OS's in the same hardware platform.

The remainder of this paper is organized as follows. Section II details our testbed. In Section III the performance evaluation methodology is described. Our results and analysis of the different tasks on both OS are presented in Section IV. We conclude the paper and discuss future work in Section V.

II. T W S N T

Our testbed is composed by Crossbow TelosB [23] motes, and two operating systems: TinyOS [19] 2.0.2 and Contiki [20] 2.3. This section gives an overview of these components.

A. Operating Systems

TinyOS [19] is an event-driven operating system for networked applications in wireless embedded systems, with a component-based architecture. It is based on the nesC programming language [24], and its core components require only 400 bytes of memory (data and instruction).

Contiki [20] is a lightweight operational system developed for constrained platforms, such as sensors nodes. Its basic configuration fits in less than 2 KiB of RAM and 40 KiB of ROM, and provides two communication stacks (μIP [25] and Rime [26]) as well as multi threading functionalities. All its modules, drivers and user applications are implemented in C programming language.

While TinyOS is sometimes considered a *de facto* standard, Contiki has a much easier learning curve for the developer, since it is based on a widely known programming language.

B. Hardware

The Crossbow TelosB [23] sensor node is a low power IEEE 802.15.4 compliant wireless platform, which includes temperature, humidity, visible light and light (including infra-red) sensors, a 16-bit 8 MHz Texas Instruments micro-controller with 10 KiB of RAM and 48 KiB of Flash Memory.

For the sensing tests, the humidity and temperature sensor SHT11 [27] was used. The communication interface is a one-wire digital connection for sending and receiving data [27]. In order to acquire data, the micro-controller sends a request containing parameters such as type (temperature or humidity), sample resolution, and control information, and then waits for the sensor to send a read command. The SHT11 transmits the acquired data along with a 1-byte CRC (Cyclic Redundancy Check). Readings can take up to $320ms$, depending on the type and resolution. The default resolution for both OS drivers is 14 bits for temperature and 12 bits for humidity, thus allowing a fair comparison of the results obtained in this work.

The TelosB platform has an IEEE 802.15.4 compliant radio transceiver, the Chipcon CC2420. The device is a low power CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)-based RF transceiver providing a maximum bandwidth of 250 Kbps when operating at the unlicensed 2.4 GHz frequency band [28]. The CC2420 is a packetizing radio, meaning that it takes the data corresponding to the payload and adds its own preamble, header and CRC. When the packet arrives at its destination, the receiver removes those fields and and checks the CRC before the data is sent to the micro-controller. The CC2420 is very flexible, allowing the implementation of a number of Media Access Control protocols on top of the basic CSMA/CA MAC[1] layer.

III. M

Our performance evaluation approach is similar to the work done by Margi et al. [29], where the key idea is to consider the elementary tasks forming the duty cycle of a WSN node: (1) sensing temperature and humidity; (2) processing data (i.e., applying security algorithms); (3) data transmission and reception.

The metrics considered in this performance evaluation are energy consumption and execution time (or duration) of a given task. In order to obtain an accurate measurement of the energy consumption, we performed direct measurements on the TelosB. We used the Agilent E3631A [30] power supply configured to provide 3.00V to power the TelosB. The Agilent 34401A [31] digital multimeter (DMM) was used to measure the current flow as the different hardware subsystems become active/inactive during the tasks' execution. Figure 1 shows a block diagram describing our measurement setup: a GPIB (General Purpose Interface Bus - IEEE 488 standard) cable was used to connect the Agilent 34401A DMM to a computer running the software LabView [32], which collects and records the measurement samples. The DMM was configured to provide a reading rate of 60 Hz.

In this measurement scenario, we ran each task considered and measured the current drained. We obtained the charge via time integration of the current and then, since voltage is constant, we obtain the energy consumption of each task. We also measured the current drained when the system was in idle (i.e., no particular tasks being executed), obtaining the threshold that is deduced from the measurements. Therefore,

[1]MAC is an acronym used both for Message Authentication Code (in the security field) and for Medium Access Control (in the networking area). Aiming to avoid confusion, we refer to MAC algorithms for the former and MAC layer when referring to the latter.

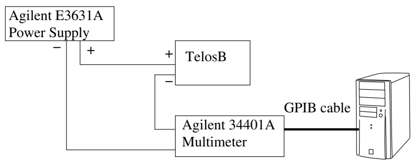

Fig. 1. Measurements' setup

the system's overall energy consumption is given by the energy consumed by each task added to the energy consumed when the system is idle.

The tasks evaluated in this work are the following:

- Sensing: humidity, temperature, and light;
- Encryption: C -2, AES and Skipjack;
- MAC calculation: CMAC and M ;
- AEAD (Authenticated-Encryption Schemes): L S and OCB;
- Communication: transmission and reception of 12 bytes of data as well as idle listening.

The algorithms for confidentiality (namely, C -2, AES and Skipjack), integrity and authenticity (namely, M - and CMAC, using the C -2 as underlying block cipher), as well as the confidentiality and integrity/authenticity algorithms (namely L S and OCB, also using the C -2), have been developed using the C language. We tested some variations on the coding in order to identify which would be the most optimized constructions for each algorithm. For example, we evaluated the impact of adopting tables with pre-computed results and tested versions of the cipher optimized for a single key size. The most efficient versions obtained from similar coding strategies are the ones considered in the benchmark.

The results presented in this work represent the average for each execution set (10 samples acquired), with 99% confidence interval.

IV. P E

In this section we present the performance evaluation for the different tasks: sensing, processing, and communication.

A. Sensing Tasks

In order to obtain readings regarding only the sensing tasks, we turned off the radio. Observe that the readings are relative to the idle energy consumption value.

Table I presents the average execution time (in milliseconds) and average energy consumption (in microjoules) for the sensing tasks running on TelosB. Regarding the humidity measurements, we observe that Contiki takes 74.6% of the time TinyOs needs to obtain a reading. However, TinyOS consumes only 24.9% of the total energy Contiki does to complete the task. Concerning the temperature readings, Contiki takes 85.6% of the time TinyOS does but, on the other hand, TinyOS consumes 24.2% of the energy Contiki does.

The two OS evaluated behave differently to obtain the sensing readings. Contiki turns on the MSP430 micro-controller analog-to-digital converter (ADC) at system initialization, and since the light sensors are two photo-diodes connected to the ADC, the cost of a light reading is very low (just a matter of reading some registers), and are not presented here because they are included in the idle value. On the other hand, TinyOS turns on the ADC converter when needed, causing the light readings to drain more current than the idle mode, and take longer time to be executed.

TABLE I
A , 99%
, T B.

	Contiki		TinyOS	
Task	**Duration** (ms)	**Energy** (μJ)	**Duration** (ms)	**Energy** (μJ)
Humidity	78 ± 0	436 ± 4	104 ± 14	108 ± 3
Temperature	232 ± 9	1370 ± 7	271 ± 6	331 ± 7
Light	–	–	59 ± 5	68 ± 8

The results shown in Table I reflect the differences between implementations from both OS core and hardware drivers. Even when hardware restrictions are the same, the OS overhead is different. TinyOS is more energy efficient than Contiki concerning sensing tasks, but Contiki takes less time than TinyOS to obtain a sensing reading.

B. Processing Tasks

Processing tasks in WSNs are usually related to the OS maintenance tasks or to the processing of sensed data. In our particular case, we decided to look at security algorithms tasks, which are usually resource demanding.

The reduced availability of resources on sensors imposes several limitations over cryptographic algorithms that can be effectively deployed in these platforms. According to the results presented by Law et al. [7], the most promising solutions to address confidentiality in such constrained platforms are AES [14] and Skipjack [15] for scenarios with, respectively, high and low security requirements. More recently, however, two versions of the C block cipher have been proposed for such scenarios [16], [17]. Roman et al. [8] present an extensive survey of existing implementations of cryptographic primitives for sensor networks, which provides performance measurements of both hardware and software implementations of ciphers, hash functions and elliptic curve algorithms. However, the work focus on processing time and memory footprint of such implementations, lacking a further evaluation of their energy consumption, a critical issue in WSNs.

For this evaluation, C -2 was configured to use 12-byte blocks and keys, AES uses 16-bytes blocks and keys, and Skipjack uses 8-byte blocks and 10-byte keys.

Table II shows the average execution time (in milliseconds) and energy consumption (in microjoules) for encryption algorithms running on TelosB. To obtain the results presented in this section, we executed 10 sets of 100 task repetitions, and then calculated the average, with 99% confidence interval. Once again, we turned off the radio to isolate the results presented here, which are relative to the idle energy consumption.

TABLE II
A , 99%
, T B.

	TinyOS		Contiki	
Task	**Duration** (*ms*)	**Energy** (μJ)	**Duration** (*ms*)	**Energy** (μJ)
C -2				
init	67 ± 10	267 ± 31	59.5 ± 0.7	323 ± 2
crypt	175 ± 8	944 ± 27	180.1 ± 0.6	1012 ± 3
decrypt	192 ± 10	1035 ± 26	200.3 ± 0.8	1128 ± 3
AES				
init	109 ± 8	526 ± 7	94 ± 1	531 ± 2
crypt	401 ± 10	2144 ± 10	381 ± 1	2080 ± 2
decrypt	476 ± 8	2562 ± 15	470 ± 1	2550 ± 3
Skipjack				
init	5 ± 1	24 ± 2	5.7 ± 0.6	20 ± 2
crypt	175 ± 10	942 ± 33	193.2 ± 0.8	1060 ± 2
decrypt	196 ± 8	1046 ± 15	205.4 ± 0.6	1127 ± 3

Notice from Table II that the ratio between execution time and energy consumption for any given OS is quite stable, as expected. Regarding the execution time and energy consumption, we observe that all algorithms perform better in TinyOS than in Contiki, except for AES's and C 's initialization, which are faster in Contiki.

The deployment of efficient Message Authentication Codes (MACs) is also an important issue in sensor networks. A reasonable strategy in such constrained scenarios is to adopt a cipher-based MAC algorithm, such as CMAC [33], [34] and M [18], in order to reduce the memory requirements for the authentication itself. Using C -2 as underlying cipher, both our CMAC and M implementations operate with 12-byte blocks and 12-byte keys, generating 4-byte tags.

Table III shows the average execution time (in milliseconds) and energy consumption (in microjoules) for these MAC algorithms running on TelosB. We observe that the MAC algorithms are faster on TinyOS but consume more energy, while Contiki takes a bit more time but spends less energy.

TABLE III
A , 99%
, MAC T B.

	TinyOS		Contiki	
Task	**Duration** (*ms*)	**Energy** (μJ)	**Duration** (*ms*)	**Energy** (μJ)
M				
init	168 ± 10	877 ± 25	250 ± 0.0	804 ± 3
getTag	218 ± 13	1142 ± 9	328.1 ± 0.0	1002 ± 7
CMAC				
init	139 ± 12	678 ± 22	234.3 ± 0.0	754 ± 12
getTag	132 ± 8	666 ± 20	250.0 ± 0.0	772 ± 8

When both confidentiality and authenticity are necessary, AEAD (Authenticated Encryption with Associated Data) schemes are usually deployed, allowing the encryption of part of the message and the authentication of the encrypted data together with some plaintext (the "associated data"). Table IV shows the average execution time (in milliseconds) and energy consumption (in microjoules) for AEAD algorithms running on TelosB. Our L S and OCB implementations rely on C -2 as underlying cipher and are both applied to 12 bytes of data, 8 bytes of associated data, generating a 4-byte tag by means of a 12-byte key. Those sizes were chosen to match the architecture model used by TinySec [35] and MiniSec [36]. The results obtained show that TinyOS is faster than Contiki, but consumes more energy.

TABLE IV
A , 99%
, AEAD T B.

	TinyOS		Contiki	
Task	**Duration** (*ms*)	**Energy** (μJ)	**Duration** (*ms*)	**Energy** (μJ)
OCB				
init	240 ± 10	1200 ± 30	340 ± 5	1079 ± 21
crypt	679 ± 18	3794 ± 23	1109.37 ± 0.00	3461 ± 4
decrypt	665 ± 8	3762 ± 35	1101 ± 10	3423 ± 7
L S				
init	170 ± 9	901 ± 30	262 ± 5	799 ± 11
crypt	598 ± 18	3445 ± 100	975 ± 6	3012 ± 14
decrypt	585 ± 10	3401 ± 3	975 ± 6	3008 ± 15

C. Communication Tasks

Communication tasks are related to transmission and reception of data packets, and depend on the communication stack used (i.e., network and MAC layer protocols).

Contiki implements two communication stacks: the μIP [25], which is compatible with the TCP/IP stack, and Rime [26]. Rime stack uses the X-MAC [37] MAC layer, which is an asynchronous low power, duty-cycled scheme (it turns the radio on and off periodically, instead of keeping it always on listening for data). On the other hand, the μIPv6 stack uses the SICSLoWMAC MAC layer protocol, which simply put the packets into standard 802.15.4 frames, and leads to higher energy consumption since the radio is always on, as observed during our experiments.

In TinyOS, the default MAC scheme for the CC2420 is the standard 802.15.4 CSMA/CA, which basically listens to the channel to detect if there is any transmission. This implies in an elevated energy consumption, since the radio will spend most of the time in idle listening. In order to minimize consumption, TinyOS provides an interface called Low Power Listening [38], which can set the radio sleep interval according to the users' needs.

In our tests, the sleep interval in TinyOS was set to 500 ms in order to approximate to the Contiki implementation of X-MAC, thus leading to a fair comparison. Also, transmission and reception used broadcast packets for both OS.

Table V presents the average execution time (in milliseconds) and energy consumption (in microjoule) for the transmission of a packet having 12-bytes of payload, both in Contiki using Rime/X-MAC and in TinyOS using the TelosB's LPL interface. Notice that the energy consumption is relative to the idle energy consumption value. We observe that TinyOS is more efficient in terms of energy consumption and execution time both for reception and transmission. Regarding transmission (resp. reception), TinyOS spent 71.8% (resp. 33.6%) of Contiki's energy, and took 76.6% (resp. 43.1%) of Contiki's time to complete the task.

TABLE V
A , 99%
, 12 .

	Contiki		TinyOS	
Task	Time (*ms*)	Energy (*mJ*)	Time (*ms*)	Energy (*mJ*)
TX 12 Bytes	828 ± 76	46 ± 4	634 ± 14	33.5 ± 0.3
RX 12 Bytes	318 ± 114	17 ± 6	137 ± 40	5.9 ± 0.1

D. Discussion

The results presented in this section show the different behaviors of the same task running on different operating systems. As shown in Section IV-A, Contiki is faster for any sensing operation, but leads to a higher energy consumption when compared to TinyOS. It is important to remember that when using Contiki, the ADC is by default turned on the whole time, sampling light measurements on both photodiodes, which causes the idle consumption on Contiki to be higher than in TinyOS. Also, this probably leads to smaller execution times.

In the case of encryption algorithms, both OS show only small differences in execution time and energy consumption. As for the tested MAC algorithms, TinyOS presented a lower execution time, but consumed more energy for the M algorithm with a small amount of authenticated data. For the AEAD algorithms, we also observed that TinyOS presented a better performance regarding execution time, but presented higher energy consumption for all algorithms evaluated.

In terms of execution time and energy consumption for communication tasks (presented in Table V), we notice that TinyOS is faster and consumes less energy, while Contiki using the Rime stack with X-MAC spends more time transmitting and receiving, which increases the overall energy consumption. Furthermore, the design and implementation decisions made throughout the OS development will impact the WSN applications in different aspects. When Contiki developers decided to add X-MAC to the Rime stack, they likely intended to make the WSN device more energy efficient. However, this characteristic could impact negatively on an application with a duty-cycle having different constraints at the application and MAC layer levels. Thus, the OS characteristics have significant impact on the overall WSN application development and performance, and the choice of an OS should be based on the specific requirements of the WSN application.

Finally, when we consider the single execution of a security task, the cost introduced is very low compared to the costs of communication. This is depicted in Figure 2, which summarizes energy consumption results presented in Tables I-V.

V. C F W

Wireless Sensor Networks (WSNs) are a valuable technology to support countless applications in different areas. Given the WSN nodes resource constrained nature, designing energy-aware applications and communication protocols is critical. The operating system running on the WSN node also interferes with the node overall behavior, and its energy consumption.

In this paper, we presented results obtained from a set of experiments aiming to measure the execution time and energy consumption of several different tasks (including sensing, communication and security algorithms execution) running on two different operating systems (namely Contiki and TinyOS) on the same hardware platform (Crossbow TelosB).

Sensing tasks run faster on Contiki, but are more energy efficient on TinyOS. Regarding processing tasks (i.e., security algorithms), the results are in general quite similar for both OS. The communication tasks perform better on TinyOS, probably due to a more efficient usage of the communication stack. Finally, the results we obtained showed us that both OS could be optimized to decrease the energy consumption, indicating that WSN developers need to be aware of this differences in behavior to improve the overall performance of their nodes.

As future work, we intend to run measurements on other hardware platforms (such as MicaZ) to understand how the same OS behaves on different platforms. Additionally, we would like to further investigate the behavior of the communication stack, and how it affects idle consumption and other tasks being executed.

VI. A

This work was supported by the Research and Development Center, Ericsson Telecomunicações S.A., Brazil.

R

[1] T. Arampatzis, J. Lygeros, and S. Manesis, "A survey of applications of wireless sensors and wireless sensor networks," in *Proc. of the 2005 IEEE International Symposium on Intelligent Control - Mediterrean Conference on Control and Automation*, June 2005, pp. 719–724.

[2] A. Alemdar and M. Ibnkahla, "Wireless sensor networks: Applications and challenges," *9th International Symposium on Signal Processing and Its Applications (ISSPA 2007)*, pp. 1–6, February 2007.

[3] D. Culler, D. Estrin, and M. Srivastava, "Overview of sensor networks," *Computer Magazine*, vol. 37, no. 8, pp. 41–49, 2004.

[4] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *WSNA '02: Proc. of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002, pp. 88–97.

[5] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," in *SenSys '05: Proc. of the 3rd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2005, pp. 51–63.

[6] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A survey of key management schemes in wireless sensor networks," *Computer Communication*, vol. 30, no. 11-12, pp. 2314–2341, 2007.

[7] Y. W. Law, J. Doumen, and P. Hartel, "Survey and benchmark of block ciphers for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 65–93, 2006.

[8] R. Roman, C. Alcaraz, and J. Lopez, "A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 231–244, 2007.

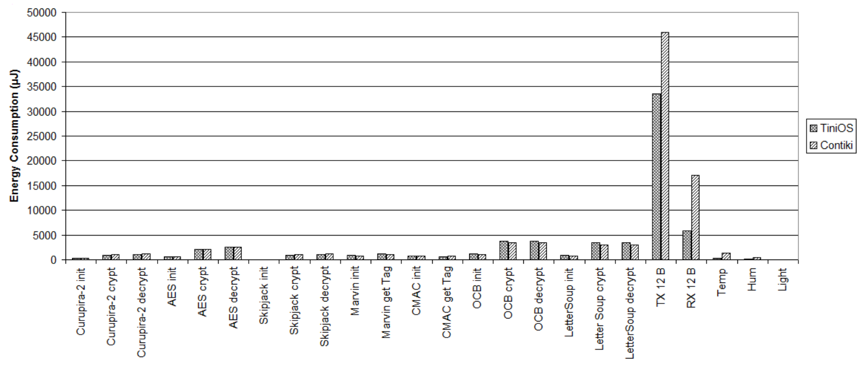

Fig. 2. Energy consumption for security tasks for TinyOS and Contiki

[9] M. Passing and F. Dressler, "Practical evaluation of the performance impact of security mechanisms in sensor networks," 2006, pp. 623–629.

[10] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, "Analyzing and modeling encryption overhead for sensor network nodes," in *WSNA '03: Proc. of the 2nd ACM international conference on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2003, pp. 151–159.

[11] L. B. Oliveira, D. F. Aranha, E. Morais, F. Daguano, J. López, and R. Dahab, "TinyTate: Computing the Tate Pairing in Resource-Constrained Sensor Nodes," in *6th IEEE International Symposium on Network Computing and Applications (NCA 2007)*, 2007, pp. 318–323.

[12] L. Oliveira, M. Scott, J. Lopez, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," June 2008, pp. 173–180.

[13] B. Doyle, S. Bell, A. Smeaton, K. McCusker, and N. O'Connor, "Security considerations and key negotiation techniques for power constrained sensor networks," *The Computer Journal*, vol. 49, no. 4, pp. 443–453, 2006.

[14] J. Daemen and V. Rijmen, *The Design of Rijndael: AES – The Advanced Encryption Standard*. Heidelberg, Germany: Springer, 2002.

[15] NSA, *Skipjack and KEA Algorithm Specifications, v2.0*, National Security Agency, 1998.

[16] P. Barreto and M. Simplicio, "C , a block cipher for constrained platforms," in *Anais do 25^o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribudos - SBRC'07*, vol. 1. SBC, 2007, pp. 61–74.

[17] M. Simplicio, P. Barreto, T. Carvalho, C. Margi, and M. Näslund, "The C -2 block cipher for constrained platforms: Specification and benchmarking," in *Proc. of the 1st International Workshop on Privacy in Location-Based Applications - 13th European Symposium on Research in Computer Security (ESORICS'2008)*, vol. 397. CEUR-WS, 2008. [Online]. Available: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-397/

[18] M. Simplicio, P. Barbuda, P. Barreto, T. Carvalho, and C. M. i, "The Marvin Message Authentication Code and the LetterSoup Authenticated Encryption Scheme," *Security and Communication Networks*, vol. 2, no. 2, pp. 165 – 180, March 2009.

[19] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.

[20] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. of the 1st IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, 2004.

[21] NIST, *Special Publication 800-38B Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*, National Institute of Standards and Technology, U.S. Department of Commerce, May 2005, http://csrc.nist.gov/publications/PubsSPs.html.

[22] T. Krovetz and P. Rogaway, "Internet draft: The OCB authenticated-encryption algorithm," http://www.cs.ucdavis.edu/~rogaway/papers/ocb-id.htm, March 2005.

[23] Crossbow, "TelosB Datasheet," http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf, 2008.

[24] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *PLDI'03: Proc. of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, NY, USA: ACM, 2003, pp. 1–11.

[25] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *MobiSys '03: Proc. of the 1st international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2003, pp. 85–98.

[26] ——, "Rime - A Lightweight Layered Communication Stack for Sensor Networks," in *Proc. of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, Jan. 2007. [Online]. Available: http://www.sics.se/~adam/dunkels07rime.pdf

[27] Sensirion, "SHT1x Datasheet," http://www.sensirion.com/en/pdf/product_information/Datasheet-humidity-sensor-SHT1x.pdf, 2009.

[28] Chipcon, "CC2420 Datasheet," http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2007.

[29] C. B. Margi, V. Petkov, K. Obraczka, and R. Manduchi, "Characterizing energy consumption in a visual sensor network testbed," in *2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2006)*, 2006.

[30] Agilent, "E363xA Series Programmable DC Power Supplies," http://cp.literature.agilent.com/litweb/pdf/5968-9726EN.pdf, 2009.

[31] ——, "Agilent 34401A Multimeter," http://cp.literature.agilent.com/litweb/pdf/5968-0162EN.pdf, 2007.

[32] NationalInstruments, "LabView," http://www.ni.com/labview/, 2009.

[33] T. Iwata and K. Kurosawa, "OMAC: One-key CBC MAC," in *Fast Software Encryption – FSE'2003*, ser. Lecture Notes in Computer Science, vol. 2887. Springer, 2003, pp. 129–153.

[34] NIST, *Special Publication 800-38B – Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, National Institute of Standards and Technology, 2005.

[35] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *2nd International Conference on Embedded Networked Sensor Systems – SenSys'2004*. Baltimore, USA: ACM, 2004, pp. 162–175.

[36] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: A secure sensor network communication architecture," in *IPSN'07: Proc. of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 479–488.

[37] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks," in *SenSys '06: Proc. of the 4th international conference on Embedded networked sensor systems*. NY, USA: ACM, 2006, pp. 307–320.

[38] D. Moss, J. Hui, and K. Klues, "Low power listening," http://www.tinyos.net/tinyos-2.1.0/doc/html/tep105.html.