Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 171 (2007) 71–81

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Rijndael for Sensor Networks: Is Speed the Main Issue?[1]

Andrea Vitaletti[2]

Dipartimento di Informatica e Sistemistica
University "La Sapienza"
Rome, Italy

Gianni Palombizio[3]

Dipartimento di Informatica e Sistemistica
University "La Sapienza"
Rome, Italy

Abstract

We present an implementation of Rijndael for wireless sensor networks running on Eyes sensor nodes. In previous works, Rijndael has not been considered a suitable encryption algorithm for sensor nodes because it is too slow and requires a large space in memory, a precious resource in this environment. Our implementation of Rijndael is smaller, from about 1/3 to 1/5 of the size of previous implementations. Furthermore, we observe that nowadays MAC and routing protocols for wireless sensor networks, exhibit latencies up to few seconds, and thus the few milliseconds required by Rijndael to encrypt a TinyOS message are negligible if compared to these latencies. For this reason, in our opinion the main focus on the implementation of encryption algorithms for wireless sensor networks should move from speed, to memory occupation and energy efficiency.

Keywords: AES implementation, wireless sensor networks, performance evaluation

[1] Supported by EU Integrated Project AEOLUS (FET-15964) and MIUR-FIRB project VICOM.

[2] Email: `Andrea.Vitaletti@dis.uniroma1.it`

[3] Email: `gpalombizio@etnoteam.it`

doi:10.1016/j.entcs.2006.11.010

1 Introduction

A wireless sensor network (WSN) is an ad-hoc wireless network made of sensor nodes which are able to monitor events (e.g. seismic activity, animals moving in a forest, enemies or intruders entering a monitored area, chemical agents), to process the sensed data and to communicate these data to a central node, the *sink*. The sink is a powerful base station which gathers data sensed in the network and either processes them or acts as gateway to other networks.

Sensor nodes are typically battery powered, making sensor networks highly energy constrained. Replacing batteries on hundreds or thousands of nodes, often deployed in inaccessible environments, is infeasible or too costly. Therefore, a key challenge in a wireless sensor networks is the reduction of energy consumption. For this reason most of the research in this field is focused on the development of energy efficient media access control (MAC) and routing algorithms.

Nevertheless sensor networks are becoming a cost-effective solution to a range of applications in critical domains. For example, after the recent terroristic events, there is a pressing need for the deployment of efficient and low-cost infrastructures for the detection of chemical or biological agents. When sensor networks are used in these security domains, besides energy efficiency, security become a strong and important requirement. Indeed these applications should not only timely detect a potential risk, but should also be protected from malicious attacks such as for example fake messages (i.e. an attacker injects malicious data which are erroneously interpreted by the system) or corrupted data (i.e. the attacker manipulate data in order to disguise the real information).

Motivation. Sensors are still some time away from actual mass fabrication and use.

Most of the current nodes, such as the Eyes node (http://www.eyes.eu.org/) or the TMote (http://www.moteiv.com/), typically run the TinyOS operating system, and are equipped with temperature and humidity sensors. They also support the installation of more advanced sensors such as microphones, accelerometers and motion sensors. However sensors for detecting biomedical data [12], explosives, radiation, chemical and biological toxins are becoming available. This paves the way for new and interesting applications. In [1] the authors study the environmental problems involving water quality and security. Clean waterways, and secure water supply are our best protection from communicable disease and the effects of chemical and biological contaminants either accidentally or intentionally released to our environment. Wireless sensor networks can be an integral part of military command, control, communications, computing, intelligence, surveillance, reconnaissance and targeting (C4ISRT) systems [2]. All the above applications have important security requirements.

Security features such as authentication, authorization and confidentiality (see [8] for an extensive discussion of the unique security challenges in Wireless Sensor Networks) can be implemented at two distinct network layers: *link layer* and *ap-*

plication layer. The use of link-layer cryptography in sensor networks is motivated by two main reasons: i) in-network processing is a common feature in most of the proposed communication protocols [15,4] and it requires intermediate nodes to access and modify the content of messages, ii) if message integrity is only checked at the final destination, the network may route packets injected by an adversary many hops before they are eventually detected. However, end-to-end security mechanisms are still useful in sensor networks and may effectively complement link-layer security. Indeed end-to-end security naturally enforces security at application level and can thus apply a suitable security level to each type of message generated by the application. Consider for example a battle damage assessment scenario. We might be interested in very high security level when reporting damages to strategic targets, but we can accept even very low security level, or no security at all, when reporting damages to insignificant targets. Observe that typically there are few important strategic targets and many insignificant targets. The ability of applying a suitable security level to each task, allow us to limit and control the overhead associated to security. This is very important in such energy constraint environment.

Nowadays security architectures for sensor networks (see for example [6,9]) implement ciphers such as RC5 or Skipjack. These algorithms can be considered secure and efficient in some environments. Nevertheless the current de facto Advanced Encryption Standard (AES) is Rjindael which is not usually adopted on sensor nodes. Indeed the limited processing and memory resources on sensor nodes, make the Rijndael implementations not efficient and thus slow in running time.

Nevertheless, time required to encipher a typical TinyOS message by Rijndael is about few milliseconds, while most of the MAC and routing protocols for sensor networks [16,14,11] exhibit latencies of few seconds for routing packets. This is because these MACs exploit a short duty cycle (i.e. the fraction of time the radio of a node is active, i.e. awake), to significantly reduce the power consumption. This difference between the few milliseconds to encipher a message and the few seconds to route the message to the sink, make Rijndael a suitable solution for sensor networks. Indeed why focus on speed when we have in any case to wait until the radio is awake to transmit a message? In our opinion, in this particular context, it would be better to focus on the optimization of Rijndael energy efficiency and memory occupation.

Our contribution. We implemented Rijndael on the Eyes sensor nodes. Rijndael is well studied and there are efficient implementations on a wide range of platforms, such as 32-bit CPUs, 64-bit CPUs, cheap 8-bit smart-card CPUs, and dedicated hardware. Nevertheless, as far as we know, there aren't specific recommendations for the implementation on 16-bit architecture, such as the Eyes node platform. Our main contributions are:

- We discuss the importance of energy efficiency and memory occupation vs. speed in the implementation of Rijndael on wireless sensor networks;
- We provide an implementation of Rijndael in software (TinyOS), that uses from about 1/3 to about 1/5 of the memory required by previous implementations;

- Our AES implementation can be exploited both for data-link encryption and for end-to-end encryption. We developed a TinyOS module which allow us to encrypt messages at application layer. This module, can possibly run on top of and effectively complement TinySec(a fully-implemented protocol for link-layer cryptography in sensor networks);
- We finally show a preliminary performance evaluation of our implementation. Our result show that AES can be effectively used for symmetric encryption, improving security at the cost of some degradation in speed.

2 Related Work

In [7] the authors present a detailed benchmark on block ciphers for wireless sensor networks. They consider security properties, storage and energy-efficiency of a set of candidates: RC5, RC6, Rijndael, MISTY1, KASUMI and Camellia. All these algorithms have been implemented and tested on Eyes sensor nodes and a ranking on the algorithms in terms of code-memory, data-memory and speed is provided. RC5 requires little code and data memory, but it is slow on speed. While RC6 excels in small code size, it is the slowest even after speed optimization. Speed-optimized Rijndael offers the highest speed but has a large code size. Indeed this implementation of Rijndael is 15842 bytes (CBC mode) for speed-optimized code, and 14716 bytes (CBC mode) for size-optimized code. Since this is almost a quarter of the flash memory of a Eyes node, this solution might be considered unpractical in many scenarios.

We observe that the performance provided in [7] are in contrast with those provided in [3]. Indeed in [3], RC5 is reported to have a speed higher than Rijndael.
All the above experiments have been performed in standalone mode, namely without interaction with the OS. Most of the ciphers code used in [7], has been taken from OpenSSL as it is, without any particular optimization for the Eyes platform. This justify a further effort in implementing a Rijndael algorithm for Eyes hardware which is smaller in size and provides reasonable speed performance.
In [5] the authors use an AES implementation, but they do not provide any performance evaluation. In [13] the authors quantify the energy cost of authentication and key exchange based on a public-key cryptography on the 8-bit ATMEL ATMEGA128L microcontroller platform. They also used an AES assembly implementation for symmetric encryption/decryption.
TinySec [6] is the first fully-implemented protocol for link-layer cryptography in sensor networks. TinySec supports two different security options: *authenticated encryption (TinySec-AE)* and *authentication only (TinySec-Auth)*. In authenticated encryption, TinySec encrypts the data payload and authenticates the packet with a message authentication code. Authentication only mode allows us to authenticate the entire packet with the TinySec message authentication code, but the data payload is not encrypted. The authors found RC5 and Skipjack ciphers to be most appropriate for software implementation on embedded microcontrollers. They wrote that Rijndael is too slow, but in a note of the paper is written: *“... AES can be im-*

plemented efficiently on our platform, with performance not much worse than RC5 and Skipjack". Unfortunately they did not provide any further information.
The paper evaluates TinySec performance by experiments. TinySec energy consumption, even when used in the most resource-intensive and most secure mode, is a modest 10%, and the low impacts on bandwidth and latency prove that software based link-layer security is a feasible solution for sensor networks. The same authors recognize, however, that end-to-end security mechanisms are still useful in sensor networks and may effectively complement TinySec.
In [3] an Intrusion-tolerant routing protocol for WSN is presented. The authors implemented both RC5 and AES. The Rijndael implementation requires about 9Kbyte, but speed performance are quite poor, about 100ms for ciphering 128 bit on a Atmega128.
Finally in [9], Perrig et al. present a suite of security building blocks optimized for resource constrained environments and wireless communication. In particular the authors present SNEP for data confidentiality and two-party data authentication, and μTESLA for authenticated broadcast.

3 Keying mechanisms

A key mechanism is designed to distribute and share cryptographic keys over the network. We can roughly identify two main keying mechanisms:
Single shared key. In this case all the nodes in the network are provided with a single shared key. Key distribution is simple, indeed a common assumption is that the shared key is loaded into nodes before deployment. However this mechanism cannot protect against tampering. In other words if an adversary can break a node and access the shared secret, she can eavesdrop on traffic and inject messages in the network.
Per-link key. In this case two nodes share a key if and only if they can communicate, namely if they are neighbors. Thus, each couple of nodes share a distinct per-link key. This implies a quite challenging key distribution mechanism, but enforce an higher security level. Indeed, even breaking a node, an adversary can only eavesdrop traffic directed to the broken node and possibly inject traffic to its neighbors. An evolution of this scheme is group keying which also allows passive participation and local broadcast.

In [10] the authors describe a probabilistic model and two protocols to establish a secure pair-wise communication channel between any pair of sensors in the wireless sensor network, by assigning a small set of random keys to each sensor.

In our work we used a simple extension of the single shared key mechanism. Indeed we load a shared master key on the nodes. This master key is used to exchange a session key, which is then used to encrypt end-to-end communications. The validity of the key is limited to a single session or in some circumstances to a fraction of the length of the session, depending on the level of security required.

4 End-to-end security with Rijndael

In this paper we present an end-to-end security system based on Rijndael. Our system can run on top of TinySec; at the moment, we demand message authentication and integrity, to the underlay link-layer protocol. We stress that while TinySec is fully implemented, our work is focused on ciphering messages at application level with Rijndael and generating and exchanging keys.

4.1 Ciphering messages

In the previous sections, we have motivated the choice of Rijndael as our reference encryption algorithm. Rijndael guarantees high security, but in order to build a secure system, we must prevent adversaries from learning even partial information from an enciphered message. The *initialization vector (IV)* and the *mode of operation*, have a great impact on the quality of a secure system. In the following, we briefly discuss the use of these two elements in our system.

Initialization Vector. Initialization vectors are used to prevent two identical sequence of text from producing the same exact ciphertext when encrypted. Since the receiver must know the IV to decrypt a message, the security of most encryption schemes do not rely on IVs being secret. We thus chose to transmit a 16 bits initialization vector in clear. Since the IV must be communicated to the receiver, a longer IV would require to reduce the available space for the payload, which is already quite small (see section 4.2 for further details). For this reason we assume 16 bits a reasonable tradeoff between security requirements - a long IV, means high security - and space reserved for data transmission - a small IV subtracts less space to the payload. Since AES requires an IV of 128 bits, the remaining bits are padded by the receiver. In any case, this tradeoff can be redefined favoring security or space for data.

Mode of operation. A block cipher, such as Rijndael, is a keyed pseudorandom permutation over small bit strings, typically 8 or 16 bytes. Since we usually want to encrypt messages longer than 8 or 16 bytes, block ciphers require a mode of operation to encrypt longer messages. A mode of operation breaks the original message in blocks of a suitable size, and thus allows the algorithm to cipher block by block. Cipher Block Chaining (CBC) is an operation mode which leaks only a small amount of information in the presence of repeated IVs and it is provably secure when IVs do not repeat. In most WSN scenarios, a typical data message frequency is few messages per minute, and thus the time to exhaust all the possible 2^{16} initialization vectors is quite long. When all the IVs are exhausted a new session key must be generated.

A k-byte block cipher in CBC mode produces ciphertexts whose lengths are multiples of k, in particular ciphering a message of length $k + 1$ results in a $2k$ length ciphertext. Since the energy required to transmit a message is proportional to the size of the message, we would like ciphertext and plaintext to have the same length. For this reason we used Cipher Text Stealing (CTS) mode of operation. The CTS mode behaves like the CBC mode for all, but it produces ciphertext whose length matches the plaintext length.

4.2 Message structure

TinySec defines two types of messages, depending on the security features required by the communication: *only authentication messages* (TinySec-Auth packet) or *authentication and cryptography messages* (TinySec-AE packet). The TinySec-Auth packet imposes some modifications to the standard TinyOS packet: the group and CRC fields are removed and the Message Authentication Code (MAC) field is introduced to authenticate the message. The TinySec-AE packet, besides the MAC field, introduces two new fields named *Src* and *Ctr*. They are both used as variables to define the initialization vector.
Hence TinySec packets, require a modification of the structure of the TinyOS packets. Indeed MAC, Src and Ctr fields have to be added, and the group and CRC fields are no longer required. For this reason, TinySec packets are slightly longer than TinyOS ones and thus in general require more energy for transmission.
On the contrary, in our solution we do not modify the message size. We use a standard TinyOS packet, just using two bytes of the payload to store the IV (16 bits). This solution has the benefit of minimizing the impact on the communication layer. Indeed all the operation on the IV are performed at application layer. Furthermore observe that since we only work on the payload, we are fully compliant with TinySec.

4.3 AES as a TinyOs Module

Rijndael is well studied and there are efficient implementations on a wide range of platforms, such as 32-bit CPUs, 64-bit CPUs, cheap 8-bit smart-card CPUs, and dedicated hardware. Nevertheless, as far as we know, there aren't specific recommendations for the implementation on 16-bit architecture, such as the Eyes node platform. We implemented Rijndael in NESC, a C dialect which is the reference development language on TinyOS. We based our 16-bit implementation on an 8-bit reference implementation. We mainly focused on space optimization. In particular we reduced the number of tables used to speed-up the computations in the Galois field. Obviously the gain in space implies an increase in computational time. We also carefully designed the code such that encryption and decryption functions share as much code as possible. Finally we adopted few empirical "implementation tricks". For example we experienced a degradation in performance when *nested for* were used, thus we replaced the inner *for* with the equivalent set of instructions.

AES cryptographic features are implemented in a new TinyOS module dubbed *AES*. In figure 1 we show how the AES configuration module (AESC) is connected to the modules of a typical wireless communication application.

Our effort was in designing a simple and intuitive system architecture. Programmers need only to work on the init function, to initialize the cryptographic

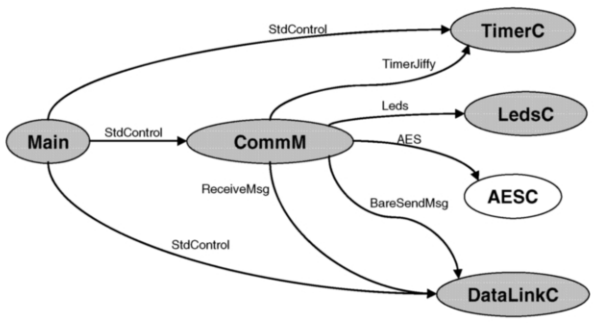

Fig. 1. AESC is the module required to add AES cryptographic features.

	CPU (bit/MHz)	RAM (Byte)	FLASH (KByte)
MSP430	16/1	2048	60
ATMEGA103	8/4	4096	128
ATMEGA128L	8/8	4096	128

Table 1
hardware

module, and on the send and receive functions to encrypt/send a message and receive/decrypt it.

Init. The `StdControl.init()` function calls the `AES.KeyExpand(Key,128)` function that initializes the cryptographic module and generates the round keys from the session `Key`.

Send. `SendMsg()` calls the `AES.CTS.encode(msg.data,msg.length,seed)` function. This function encrypts the message data using the `seed` as initialization vector. The seed is placed in the last 2 bytes of the payload of the encrypted message. Observe that AES requires an initialization vector of 128 bits. For this reason, the last bits of the seed are padded.

Receive The reception event `ReceivevMesg.receive(TOS_MsgPtr m)` calls `AES.CTS.decode(m->data,m->length,seed)` which decrypts the incoming message. The seed is obtained from the last two bytes of the received encrypted payload.

5 Performance evaluation

Although the standard choice when a block cipher is required is either AES or Triple-DES, security architectures for sensor networks [6,9] implement ciphers such as RC5 or Skipjack. These algorithms are assumed to be more suitable for software implementation in embedded microcontrollers. In particular there are two main concerns about the implementation of Rijndael on sensor nodes: it is often considered too slow and it requires more space in memory, if compared to RC5. The work by Law et al. [7] prove that Rijndael can be quite fast.

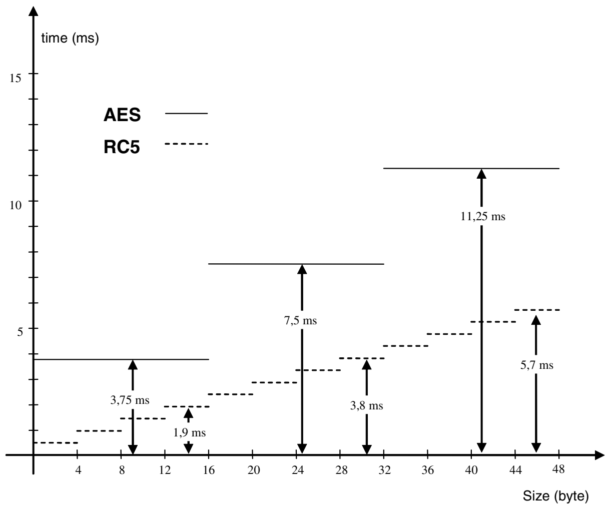

Fig. 2. AES performance.

In any case we observe that most of the MAC and routing protocols for sensor networks [16,14,11] exhibit latencies of few seconds for routing packets. This is because these MACs exploit a short duty cycle (i.e. the fraction of time the radio of a node is awake), to reduce power consumption. Time required to encipher a typical TinyOS message by Rijndael is about few milliseconds.

In our opinion, this difference between the few milliseconds to encipher a message and the few seconds to route the message to the sink, make Rijndael a suitable solution for sensor networks. Indeed, Rijndael speed although important, might be considered a secondary issue in an environment with the above latencies. On the contrary, wireless sensor nodes require to focus on implementations of Rijndael which are small in memory and energy efficient.

In figure 2 we show the results of our experiments. The figure shows the time required to cipher messages of different sizes both with AES and RC5. The performance of our implementation of RC5 are comparable with those provided in TinySec, even if we have to stress that a fair comparison is quite hard. Indeed our implementation runs on a MSP430F149 (4 MHz 16 bit) while TinySec runs on a Atmega128L (8 MHz 8 bit), see table 1. Our implementation of AES encrypts 128 bit, with a key of 128 bit in 3.75ms, while RC5 requires only 1.9ms, with a key of 128 bit, 12 rounds and blocks of 32 bits. We recall that the AES implementation in [7] is about 15Kbytes in memory size and encryption speed of Rijndael is even better than RC5, while the AES implementation in [3] is about 9Kbytes, but speed performance are quite poor. Our AES implementation requires only 241 bytes in RAM and 3322 bytes in ROM. Hence, the size of our implementation of Rijndael is about 1/3 of that in [3] and about 1/5 of that in [7].

Furthermore our current implementation of Rijndael allows us to encrypt with key of 128, 192 and 256 bits. If we limit the length of the key to 128 bits, we only need 177 bytes in RAM.

6 Conclusion and future work

In our opinion, en-decryption speed is not the main constraint for a secure wireless sensor network scenario. Indeed as long as latency of the WSN communication protocols will be so high (few seconds), we will be allowed to spend few millisecond to encipher a message with the AES algorithm. In this work we have presented an implementation of Rijndael smaller in size than previous implementations (from about 1/3 to about 1/5). However our AES implementation shows reasonable speed performance (slower than RC5 by a factor 2). We plan to extend our work in order to define a fully implemented end-to-end solution and to investigate the energy efficiency of Rijndael.

References

[1] A. Ailamaki, C. Faloutos, P. S. Fischbeck, M. J. Small, and J. VanBriesen. An environmental sensor network to determine drinking water quality and security. *SIGMOD Rec.*, 32(4):47–52, 2003.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, March 2002.

[3] J. Deng, R. Han, and S. Mishra. A performance evaluation of intrusiontolerant routing in wireless sensor networks. In *IPSN '03: Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks*, pages 349–364, 2003.

[4] J. Deng, R. Han, and S. Mishra. Security support for in-network processing in wireless sensor networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 83–93, New York, NY, USA, 2003. ACM Press.

[5] B. Dutertre, S. Cheung, and J. Levy. Lightweight key management in wireless sensor networks by leveraging initial trust. Technical Report SRI-SDL-04-02, SRI International, 2004.

[6] C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175, New York, NY, USA, 2004. ACM Press.

[7] Y. W. Law, J. M. Doumen, and P. H. Hartel. Benchmarking block ciphers for wireless sensor networks (extended abstract). In *1st IEEE Int. Conf. on Mobile Ad-hoc and Sensor Systems (MASS)*, page electronic edition, Fort Lauderdale, Florida, Oct 2004. IEEE Computer Society Press, Los Alamitos, California.

[8] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.

[9] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. Spins: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002.

[10] R. D. Pietro, L. V. Mancini, and A. Mei. Random key-assignment for secure wireless sensor networks. In *SASN '03: Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, pages 62–71, New York, NY, USA, 2003. ACM Press.

[11] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 181–192, New York, NY, USA, 2003. ACM Press.

[12] L. Schwiebert, S. K. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165, New York, NY, USA, 2001. ACM Press.

[13] A. S. Wander, N. Gura, H. Eberle, V. Gupta and S. C. Shantz Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks. In *Percom*, pages 324–328. IEEE Computer Society.

[14] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 171–180, New York, NY, USA, 2003. ACM Press.

[15] A. Woo, S. Madden, and R. Govindan. Networking support for query processing in sensor networks. *Commun. ACM*, 47(6):47–52, 2004.

[16] W. Ye, J. S. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *INFOCOM*, 2002.