A Wireless Sensor Network for Monitoring the Sound of Animals in Tropical Forests

Ilan S. Correa, Lilian C. Freitas, João Crisóstomo Costa and Aldebaro Klautau
Sensors and Embedded Systems Laboratory (LASSE) - Federal University of Pará (UFPa)
CP 8619 - Rua Augusto Correa 01 - CEP 66075-110 - Bélem - Pará - Brazil.
Web: www.lasse.ufpa.br. E-mails: {ilan, liliancf, jwey, aldebaro}@ufpa.br

***Abstract*—Wireless sensor networks has been growing and brings with it many alternatives which facilitates and turn possible some tasks, like monitoring places which access is not so easy, it is possible joining small sensors with wireless technology. The sensors to this tasks should not change the behavior of the object being observed, so it must be small which implies some hardware limitations. This work presents a implementation of wireless sensor network application for monitoring sounds of animals in tropical forest. It uses a limited radio and CPU and a little memory, and still have to manage these limitations and at same time enables the capture, storage and transmission of sounds, with the least possible delay. It was used a compression algorithm and presented an analysis that considers the radio and CPU usage, and the quality of the sound after decompression.**

1. Introduction

The analysis of sounds produced by animals, known as bioacoustics [1], enables identifying and monitoring species, estimates biodiversity of certain place, besides facilitating its study.

The great challenge on monitoring sounds of animals in forest environments, is the difficulty of access, because the forests are extremely closed and, many times, crossed by rivers (as the Amazon rainforest). These characteristics make it difficult for researchers have access. Furthermore, the monitoring of animal sounds should occur without interference on their habits.

Thereby, the use of Wireless Sensor Network (WSN) shown very promising, because it do not need a infrastructure to work (ad hoc network), do not needs people to perform sounds capturing and moreover do not interfere the natural habits of interest animal.

Howsoever, sound monitoring application requires high bandwidth, impose hard restrictions on delay and requires coordination between sensor nodes. These restrictions need efficient algorithms implemented on WSN, because such networks are usually composed of sensor nodes with little energy autonomy, with 8 or 16 bits microcontroller and its radio has low transmission rates.

The goal of this work is to describe the implementation of a WSN for monitoring sounds of animals in forests. A general purpose compression algorithm called DPCM (differential pulse code modulation) [2] was employed to allow transmitting more information on the radio link. This work is part of a project that aims at constructing a "intellectual property" (IP) core called ListenU within the NAMITEC activities [3]. This work is important in this context because it allows to test these algorithms in practice, using the testbed that is described in the sequel.

The remainder of this paper is organized as follows: the section 2 described the sensor platform used, the section 3 describes in details how the application works, the section 4 shows the result and the section 5 presents the conclusions of the work.

2. Platform and Applications

2.1. Sensor Nodes

In this work was used the sensor nodes Micaz of Crossbow Technology Inc. [4] equipped with processor ATMega128L [5] of 7.3728 MHz with 128 kB of program memory (flash), 4 kB of data memory (sram). This platform still has 512 kB of external memory (flash) for reading of measurements of sensor nodes and CC2420 [6] radio module running at 2400 MHz capable to offer a bandwidth total of 250 kbps.

An expansion slot accommodates a variety of sensing cards such as light, temperature, magnetic field, sound, and so on. The Micaz motes execute the TinyOS operating system [7] that has a programming model based on components, provided by the network embedded systems C (nesC) language [8], [9].

In this work was used the MTS300 [4] sensorboard connected on expansion slot. The microphone provided by this sensorboard has a 10 bits A/D converter, and was used a 16 bit variable to avoid overflow.

2.2. Software ListenU

As mentioned, this work is part of a project that aims at building an IP core called ListenU. This IP core will be written in SystemC and then validated. But for testing the software with Micaz nodes, the software will be first written in nesC and Java, as described in the sequel.

The ListenU application developed on this work allows capturing sound samples on sensor network and delivers packets with samples to the computer. It was divided into three modules:

- **Sensor node module**: this module is responsible capturing the sounds of animals, compress the samples, packing and transmitting samples to Sink module. This module was developed using nesC Language.
- **Sink module**: this module is a bridge between the WSN and the computer. Its works forwarding received signals

by serial port to the WSN and received packets from WSN to the computer. This module was developed using nesC Language.

- **Interface module**: it was developed in Java using the TinyOS Java API, and was based on an example application that comes with TinyOS called *Antitheft*. Is important to highlight that this module also uses TinyOS API.

The Figure 1 shows the modules distribuition on WSN, and diagram 2 illustrates communication flow.

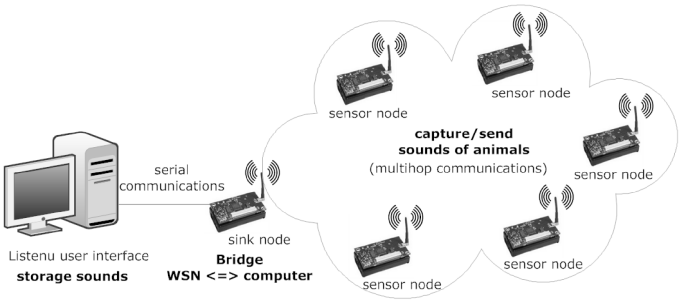

Fig. 1. The architecture of the application.

3. System Design

3.1. Data Acquisition

The ListenU application was designed to sample every time that user sends signal via Interface module, to turn possible the sampling operation, the Sensor node module maintains a buffer in RAM, that temporally stores all samples, before being compressed and transmitted. Sampling frequency and the buffer's size are pre-defined on a header file.

Each time the signal arrives on Sensor node module, the buffer is filled, when this is done TinyOS sinalizes, both operations uses *ReadStream* interface [10].

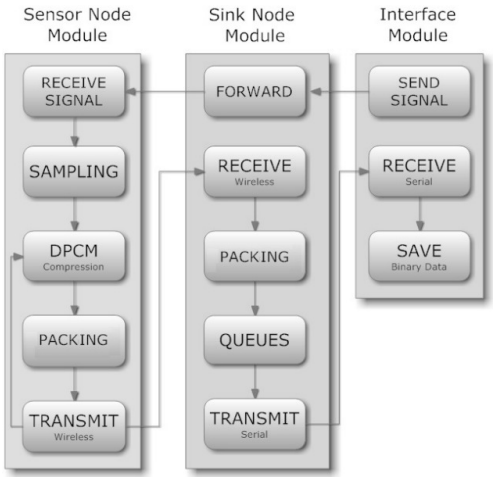

Fig. 2. The operation diagram of the system.

3.2. Communication Patterns

Two levels of communications were implemented. The computer-Sink serial communication using RS-232, and Sink-Sensor and Sensor-Sensor using multihop wireless communications. Figure 2 shows the communication flow.

The serial communication works on Sink module and in Interface module, using 57.6 kbps transmission rate, which is the highest transmission rate supported by Micaz. The serial communication on Sink uses *AMSend* and *Receive* interfaces from TinyOS.

In serial communication a latter emphasized factor is the difference between radio and serial transmission rate (250 kbps and 57.6 kbps respectively), this may causes packet loss. Therefore a queue was added on Sink module in forwarding operation, and each received packet should be queued before to be sent to computer.

The second communication pattern was wireless with multihop, which allows a node transmits a packet using multiple stage until reaches the destination.

Sink module forwards a start read signal using the *DisseminationUpdate* interface [11]. Thus the signal is delivered to each node on WSN by *DisseminationValue* interface, it reports that a signal has arrived. This operation runs with the Trickle algorithm [11], [12], which ensures that all reachable node receive the signal. Indeed the signal is a shared variable, provided by the algorithm where each node can get and set its value, but here this resource only signals, therefore only sink can set.

To collect the samples with Sink module, was used *Receive* interface with *CollectionC* component [13], it provides ways to collect all packets from the network using multihop. This component uses CTP (collection tree protocol) protocol [14], where a node is elected root of a collection tree (Sink module) and every transmitted packet should reaches the root. If a node cannot reach the root with a single jump, CTP uses multiples jumps (multihop) to ensure that all samples reaches the root.

A Sensor node which has done sampling, should send its samples to collect more samples, but if several Sensor nodes simultaneously transmits may occur network congestion. Any congestion should be avoided, thus were used *DisseminationUpdate* and *DisseminationValue* interfaces (this time using the shared variable).

Before sending a Sensor node first checks if it can sends, using *DisseminationValue*, that returns a pointer to a bool variable, it represents the transmission operation status (busy or free). Thus Sensor node knows if any node is sending, then it changes the variable's value using *DisseminationValue* interface that also reports all Sensor nodes. Now it has exclusive access to transmit its samples, thus Sensor node do it quietly. When sending is over the node change again variable's value to report that it finished the operation.

3.3. DPCM Implementation

In the WSN Sensor nodes samples and communicates with Sink module always it completes sampling, this operation produces a great number of data, and its transmission can

take much time. Thereby increasing the number of transmitted samples per packet became necessary, and was implemented the compression algorithm DPCM (Differential Pulse Code Modulation) [2] on Sensor node modules.

The DPCM algorithm increases the volume of data per packet, this is possible because is sent the difference of sample and its prediction instead the sample. Insofar the sample consumes two bytes and the algorithm produces a difference that consumes one byte. The nesC code below shows the DPCM's implementation.

```
for(i=0;i<N_SAMPLES;i+=PACKET_SIZE){
    s_rec_n_menos_2 = 0;
//reconstructed signals
    s_rec_n_menos_1 = 0;
    for(j=0;j<PACKET_SIZE;j++){
        x_ch= (s_rec_n_menos_2 + s_rec_n_menos_1)/2;
        //prediction of sample
        d = b[j+i]-x_ch; // d = the difference
        // b = buffer of sample
        //Quantization block
        if ( d < -128 )
            dq=-128; //dq = quantized difference
        else if ( d > 127 )
            dq=127;
        else
            dq=d;
        s_rec_n_menos_2 = s_rec_n_menos_1;
        s_rec_n_menos_1 = dq+x_ch;
        s->sample[j] = dq; }} //packing the differences
```

The first step of this algorithm is to calculate a prediction of a sample, through average of the prediction of two previous samples. Then is calculated the difference of between a sample and its prediction. This difference is quantized in the range from -128 to 127 (represented by 8 bits). The quantized difference is transmitted to the Sink module.

4. Results

This section presents an analysis made over the two implementation of the ListenU application. The first implementation does not use compression and the second uses the DPCM algorithm. The analysis considered the radio usage, which is presented in Table I.

TABLE I
RADIO USAGE: THE SIMULATION CONSIDERED THE TRANSMISSION OF 1000 SAMPLES.

	Transmission time	Power consumption	Assembly instructions
Original	0.8 s	33.408 mJ	61,957
Compressed	0.4 s	16.704 mJ	61,881

Table I shows some data acquired on each version of ListenU, these data were acquired based on estimation, that considers capacity and consumption values obtained from CC2420 datasheet [6]. It shows radio data considering raw samples (no compression) and DCPM compressed samples. It is observable that the implementation of algorithm introduces few instructions, this way the algorithm increases minimally the computational complexity.

Figure 3 shows another important comparison, it indicates slight difference between them. Tests were done with real signal, to assess this difference and appeared a little distortion in compressed version, nevertheless it does not affects the sound understanding.

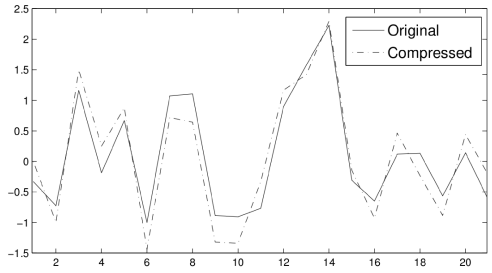

Fig. 3. A signal and its version after compression and decompression.

5. Conclusions

This work presented a complete application for monitoring, from the sampling module up to the computer interface that manipulates the received data. Some challenges had to be overcome like hardware limitations, the volume of produced data and the difficulty to testing the application especially when using Micaz nodes.

To bypass these limitations, a especial attention was given to the DPCM algorithm, which allowed energy and time saving with a small increase in computational complexity. Also the limited range achieved by the radio module was circumvented by the adoption of a multihop topology, which allows to obtain data from nodes far from the sink node. The overall performance of the system depends on both the coding (in this case DPCM) and multihop topology. In spite of focusing in sound, the developed system can be adapted to other uses.

References

[1] Thomas Rossing, *Springer Handbook of Acoustics*, Springer.
[2] M. D. Paez and T. H. Glisson, "Minimum mean-squared-error quantization in speech pcm and dpcm systems," *IEEE Trans. on Communications*, vol. 20, pp. 225–30, 4 1972.
[3] "Instituto Nacional de Ciência e Tecnologia de Sistemas Micro e Nanoeletrônicos - NAMITEC," http://namitec.cti.gov.br.
[4] "Crossbow technology inc.," http://www.xbow.com/.
[5] "Atmel inc.," http://www.atmel.com/.
[6] "Chipcon inc.," http://www.chipcon.com/.
[7] "Tinyos," http://www.tinyos.net/.
[8] David Culler Eric Brewer David Gay, Philip Levis, *nesC 1.1 Language Reference Manual*, http://nescc.sourceforge.net/papers/nesc-ref.pdf.
[9] UC Berkeley WEBS Project, "nesc: A programming language for deeply networked systems," Accessed in 2010., http://nescc.sourceforge.net/.
[10] TinyOS, "Tinyos source code documentation," http://www.tinyos.net/tinyos-2.1.0/doc/nesdoc/micaz.
[11] "Dissemination of small values," http://www.tinyos.net/tinyos-2.1.0/doc/html/tep118.html.
[12] Neil Patel David Culler Philip Levis and Scott Shenker, "Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks," *In Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2004.
[13] "Collection," http://www.tinyos.net/tinyos-2.1.0/doc/html/tep119.html.
[14] "Collection tree protocol," http://www.tinyos.net/tinyos-2.1.0/doc/html/tep123.html.