Chapter 4

Experimental results

In this section, an implementation of the algorithm previously presented is carried out. The section is divided in three different subsections. In Section 4.1 a brief introduction to the sensors is given, followed by an explanation of the experimental setup in 4.2 to finally present the results and conclusions in Section 4.3.

4.1 Sensor description

The experiment was run onto Moteiv's popular mote: Tmote Sky. Tmote Sky is an ultra low power wireless module for use in sensor networks, monitoring applications and rapid application prototyping, being a natural replacement for Moteiv's previous product (Telos).

The module was designed to fit the size of two AA batteries from which is powered. Although 2.1 to 3.6V DC cells are explicitly requested in the datasheet ([26]) in the experiment 1.5V DC cells were used, as are the ones provided along the motes. The sensors can also be powered from the USB port of a computer (as is the case of the data-gathering node). In this case, it is not necessary to use batteries.

Also, Telos module has been designed to provide a very low power operation, for what, between many other things, uses an ultra low power Texas Instruments microcontroller (TI MSP430 F1611) featuring 10kB of RAM and

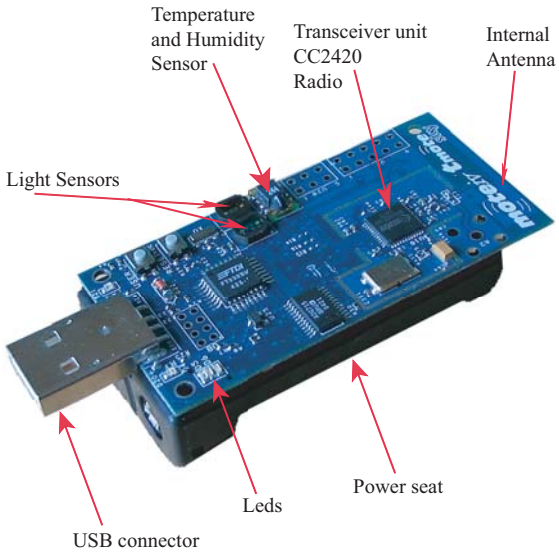

Figure 4.1: *Graphic display of the sensors used to run the experiment.*

48kB of flash. The MSP430 has an internal digitally controlled oscillator (DCO) that may operate up to 8MHz.

To be able to communicate with a PC through the USB port, Telos uses a USB controller from FTDI which, of course, requires a previous installation of FTDI's drivers on the host. Furthermore, Windows users will need the Virtual Com Port (VCP) drivers. These drivers are included on the Moteiv CD shipped with your order or downloaded from FTDI's website.

On the radio interface, Telos features the Chipcon CC2420 radio for wireless communications. The CC2420 is an IEEE 802.15.4 compliant radio, being highly configurable. Two antennas options are provided, an internal antenna built into the module and an external SMA connector for connecting to external antennas. By default, Telos is shipped with the internal antenna enabled. Although not a perfect omnidirectional pattern, the antenna may attain 50-meter range indoors and upwards of 125-meter range outdoors.

There are 4 optional sensors supported onboard:

- The TSR (*Total Solar Radiation*) and PAR (*Photosynthetically Active Radiation*) sensors measure using the microcontroller's 12-bit ADC with $Vref = 1.5V$. The photodiodes create a current through a $100k\Omega$ resistor. To calculate this current we can use Ohm's Law:

$$I = V_{sensor}/100k\Omega \tag{4.1}$$

  where V_{sensor} can be obtained as:

$$V_{sensor} = value_{ADC}/4096 \cdot Vref \tag{4.2}$$

  The Moteiv datasheet [26] includes curves for converting the photodiode's current into light values (Lux)

- Humidity and Temperature sensors are located in the external Sensirion sensor. Their readings can be converted to IS units as follows:

  For temperature, the 14 bits value returned can be converted to Celsius degrees as:

$$temperature = -39.60 + 0.01SOt \tag{4.3}$$

  where SOt is the raw output of the sensor.

  Humidity is a 12-bit value that is not temperature compensated.

$$humidity = -4 + 0.0405SOrh + (-2.8 \cdot 10^{-6})(SOrh^2) \tag{4.4}$$

  where, same as before, $SOrh$ is the raw output of the relative humidity sensor. Using this calculation and the temperature measurement, you can correct the humidity measurement with temperature compensation:

$$humidity_{true} = (Tc - 25)(0.01 + 0.00008SOrh) + humidity \tag{4.5}$$

  where Tc is the temperature measured in Celsius from equation (4.3), $SOrh$ is the raw output of the relative humidity sensor, and humidity is the uncompensated value calculated in equation (4.4).

4.2 Experimental setup

Several important variables have to be defined before we are able to run the experiment of distributed source coding.

The first one is to decide how the WSN is deployed. Is straightforward to choose the network architecture, since it is requested by the algorithm itself to be as shown in Fig: 1.1. Furthermore, we choose the network to be composed by five sensors to be able to compare results with those presented in [3].

Since the data-gathering node is represented by a sensor plugged in the USB port of the PC where the algorithm is run, we have now to decide which interface are we going to choose for reading and writing to the USB port and perform the necessary calculations. Here we have several possible solutions being the most typical ones to write a suitable code in Java or C. However, we decided to interface the motes with MATLAB. The main reason being that MATLAB is a well known platform providing a complete support for matrix processing and where we can reuse code for subsequent simulations (see section: 4.4). We have provided a description on how to interface MATLAB with TinyOS in Appendix B.

The experiment took place in the laboratory of the Automatic Control Group (School of Electrical Engineering) at KTH. In Fig. 4.2 the location of each sensor is shown. Each star denotes the position of the sensor with network address the displayed number. Our application has been developed on top of a multihop protocol so that direct line of sight is not required. The routing protocol will build the routing tree having node with network address zero as root, what means that all packets will be forwarded to this node. Hence, the node zero (we will denote the sensors by their network addresses) will be the one in charge for requesting data to a concrete node and receiving and processing the correspondent answer. The computer to which the Base Station (BS) is attached, will be the one responsible for tracking the correlation structure and determining when and which sensor must be enquired.

In the experiment we will test the behavior of the algorithm subject to two

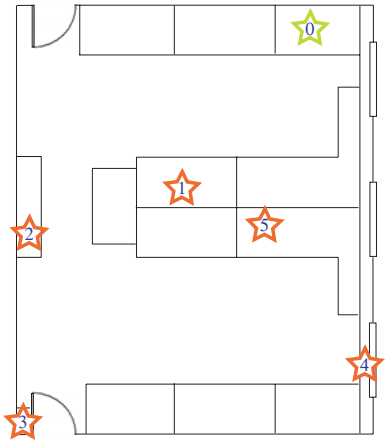

Figure 4.2: *Physical disposition of the sensors in the lab.*

different environments: indoor and outdoors. The latter one was achieved by opening the windows.

When computing the prediction for a determinate sensor we use its most recent four past measures and the current value sensed by one of the remaining nodes. Mathematically, this is expressed as

$$Y_k^{(j)} = \sum_{l=1}^{4} \alpha_l X_{k-l}^{(j)} + X_k^{(m)} \tag{4.6}$$

where, obviously, $m \neq j$

To perform the experiment we chose the temperature as the magnitude to be measured. From Section 4.1 we know that the ADC returns a 14 bit value, and from Eq. (4.3) we derive that the dynamic range expands over $[-39.60, 124.23]$ ºC. But we still have to come up with the value of several important variables as the step size, the value of K (length of the initialization module), the sample time, the maximum waiting time (the time we wait after a request for concluding whether a packet has been lost or not),...

Let us start by the step size μ and K (see Eqs. (1.13)(1.19)), since they are closely related to each other. The value of μ and K will be given by the initial conditions of the coefficients' vector $\mathbf{\Gamma}_j$ and the characteristic time of the signals sensed. Several simulations over real data yielded $\mu = 2.1 \cdot 10^{-4}$ as the quasi-optimal tradeoff value between speed of convergence and

stationary error (following the original theory by Haykin [27] we initialize $\mathbf{\Gamma}_j$ as the null vector). For K the value chosen was 30.

As sample time we chose 10 seconds (between measures belonging to one self-same sensor), in other words, two sensors with consecutive network addresses are enquired with a time difference of 2 seconds.

Finally, we set the value for the waiting timer to 3 seconds and the probability of decoding error to 0.01.

4.3 Analysis of the results

With the parameters defined in the previous section, we ran the experiment during approximately one hour and a half to yield the results shown in Figs. 4.3, 4.4 and 4.5. Let us give a small insight on each one of the subplots:

- Fig. 4.3 plots the value of the sensed data and its predictions during the run of the experiment. Note that signals are perfectly tracked, committing an unnoticeable error, so that we can only appreciate 5 different signals.

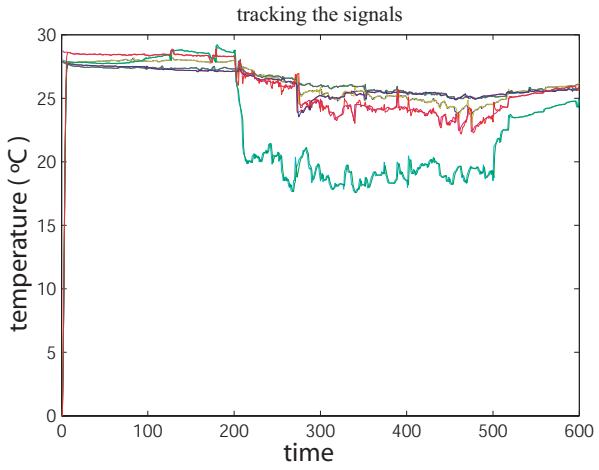

Figure 4.3: *Signals' reconstruction.*

- Fig. 4.4 shows the evolution of the error (difference between the decoded and the predicted value of the temperature) during the initialization module. It can be appreciated how it barely takes 8 samples to

track the signals with an error smaller than 0.05ºC what validates the value chosen for the step size μ, and shows that we can decrease the length of the initialization module (in other words, reduce K), with the consequent increase of the compression rate.

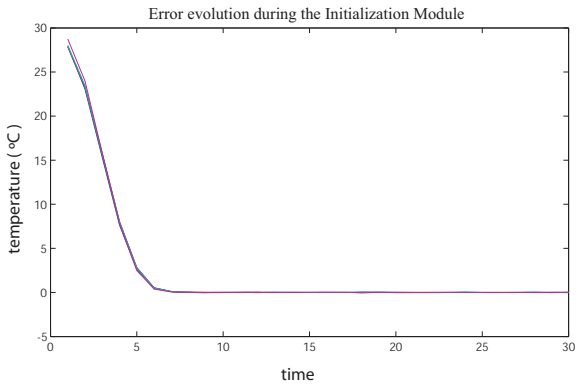

Figure 4.4: *Error evolution.*

- Fig. 4.5 is the most illustrative one when trying to show the reduction achieved in the number of bits requested by means of the algorithm. In the y-axis it shows the frequency with which a determinate number of bits (in the x-axis) has been retrieved. We can realize how, without any compression, this plot would be a single bar at value $i = 14$ with frequency 1. However, thanks to the compression algorithm we have been able to displace it to its right, being the new median of the distribution around value $i = 6$ (bits) for most of the sensors.

Now that we have a basic knowledge of what each figure means, it is mandatory to make a joint analysis of the three figures without which the potential of the compression algorithm would not be understood.

Initially the sink, where the prediction algorithm is carried out does not have any information to compute the prediction. This is the reason why at the beginning the difference between the real and the predicted values is so large (see Fig. 4.4). As the BS starts to collect measurements from several nodes, predictions become to be more and more accurate until they reach an almost perfect estimate. Once the signals have been tracked, the number of requested bits starts to decrease because the variance of the error

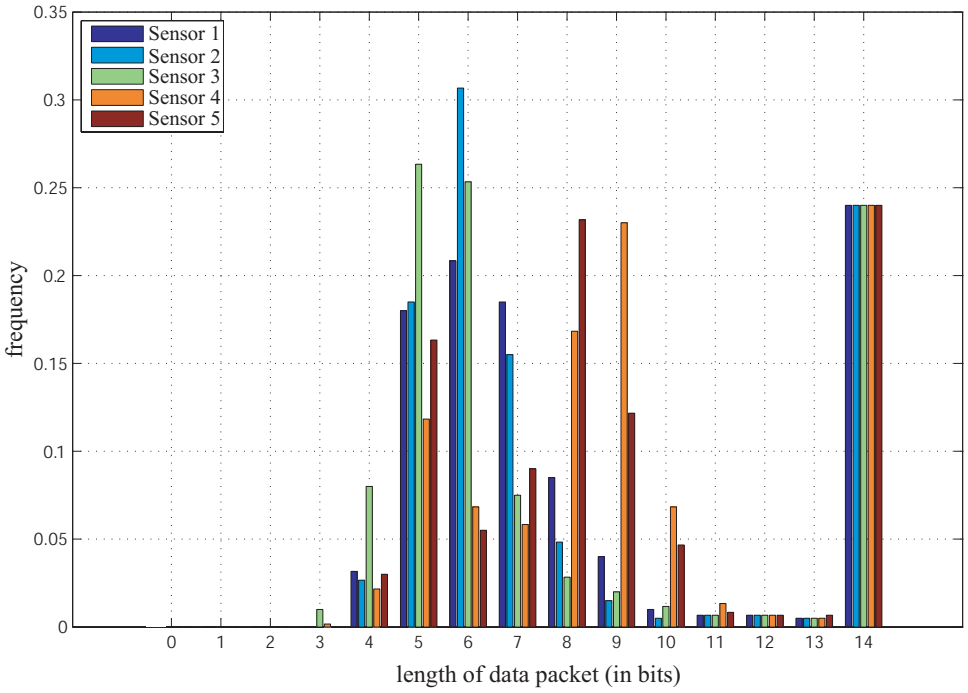

Figure 4.5: *Length of the data packets.*

decreases as a consequence of the smaller prediction error (see Eq. 1.20). In the experiment two kind of situations can be seen: the first one corresponds to an indoor environment (for all the sensors from samples K=1:200), in that moment, the window close to sensor 4 was opened (see Fig: 4.2 for more information about the location of the sensors) so that the temperature returned by this sensor (and number 5) reflects a decrease of its value at the same time that the variance becomes higher, as is typical of an outdoor environment. This can be easily appreciated in Fig. 4.3. We can see how one sensor drastically decreases its temperature at the same time the rest of sensors start to slowly decrease their sensed values of temperatures little by little(due to the slow cooling of the room). In $K = 500$ the window was closed again with the consequent stabilization and slow increase of the temperature in the room.

In Table 4.1 we show the compression rate achieved during different modules of the experiment. Note that in the table, IM stands for Initiation Mod-

ule (see Section)

Description	Case	Compression rate (%)
Whole experiment excluding IM	$k = K..600$	37.8
Whole experiment including IM	$k = 0..600$	36
All windows closed, no IM	$k = K..200$	40.08
One window opened	$k = 200..500$	34.76

Table 4.1: *Analysis of the compression rate achieved.*

Understanding Table 4.1 is of crucial importance for a deeper comprehension of the algorithm. Hence, we would like to highlight some aspects:

1. Including the Initialization Module in the computation yields a reduction in the compression rate achieved. In this way, a smaller value of K (it can be set to 10 instead of 30) will perform better.

2. Stationarity of the signals yields more precise estimates. Thus, the compression rate obtained when the windows are closed is higher than that got when one window is open. This is easily understandable: the colder external temperature provokes a descend of the temperature in the room. This change in the statistics of the temperature has as a consequence a deviation between the real value and the predictions (increase of the error), whose ultimate consequence is the increased number of bits requested.

4.4 Results of the simulation

In this section further studies on several parameters of the algorithm are presented. The objective of carrying out this simulation work is threefold: *i)* Overcome the always cumbersome and time consuming work of setting up the network, *ii)* arrive at results comparable to those of the experiment (which is unique and unrepeatable) by using the data obtained from this one, *iii)* perform analysis impossible to carry out in reality.

Simulations are performed in a similar way to that of the real experiment. The structure of the MATLAB code is basically the same as in the experiment but instead of requesting data to the correspondent node, it reads the appropriate variable in a log file. Since the data used was recorded in the experiment and we can assume that it faithfully represents the reality (the decoding error is null as it will be shown in subsection 4.4.2), we can consider this data as real.

Prior to going farther we should check the validity of the results given by the simulator. To do this we just need to introduce the measures recorded in the experiment into the simulator, which returns a compression ratio of a 36%, value which fits in that attained in the real implementation.

Once the use of simulations has been motivated and validated, we can proceed to study the effect of several parameters on the compression rate and the robustness to errors of the compression algorithm.

4.4.1 Effect of K and the number of sensors

Let us start our analysis with a parameter which obviously affects the overall compression rate: the length of the initialization module, K.

During the initialization module (IM) the sensors are asked to send their data uncompressed. Thus, while IM is taking place, compression is not being carried out. In Table 4.2 we show the compression rate achieved for three different runs.

K	Compression rate (%)
50	34.5
30	36
10	37.4

Table 4.2: *Varying the value of K.*

Simulations confirm what we already knew: the higher the value of K, the smaller the compression rate. It should also be noted that the influence of K in the final compression gain also depends on the relative value of K

respect to the length of the simulation run (this can be appreciated in Table 4.3 if we consider one individual row).

An analysis of how the number of nodes affects the overall performance was also carried out. The results have been displayed in Table 4.3, showing that larger networks attain a better performance than smaller ones. Thus, if we compare the compression gain for a WSN composed by 5 sensors and another one having 200 nodes, we can easily see an improvement in the compression gain of almost the 25%.

	number of measures				
number of sensors	600	5000	10000	30000	100000
5	37.37	37.93	37.97	37.99	38
20	44.38	45.04	45.09	45.12	45.13
50	45.78	46.47	46.51	46.54	46.55
100	46.25	46.94	46.98	47.02	47.03
200	46.48	47.18	47.22	47.26	47.27

Table 4.3: *Optimizing the compression gain.*

In Table 4.3, we assumed $K = 10$, and the IM was included for the calculation of the compression gain. For computing the presented values we considered the same distribution for the number of requested bits as the one in the experiment.

4.4.2 Robustness to errors

In this section we check the robustness of the compression algorithm against the two kinds of errors that can appear. The first type of error is a packet loss. The second is a decoding error, which means that we decoded the prediction to the erroneous codeword in the codebook. Each one of these errors will be considered in the following items:

Packet loss There are two possible ways in which a packet loss can occur: malfunction of the temperature sensing device or transmission loss. In principle, a packet loss could enormously affect the decoding

and the correlation tracking processes, because this error propagates along time (affecting those estimates that depend on this measurement). There are two possible policies to follow: use the prediction on behalf of the sensed value or simply use the last correctly decoded value for that sensor. We chose the latter one. For the simulations we considered a bursty noise channel: the Gilbert-Elliott Channel model [28], which is characterized by two states, the *Good* and the *Bad* state, denoted by *G* and *B*. Let us express the probabilities of being in each of these states as π_G and π_B. The wireless channel is modelled by choosing the model parameters to match a concrete probability of packet loss and the average burst length. The Gilbert-Elliott model has been depicted in Fig. 4.6, where p_{ij} $(i, j \in \{g, b\})$ is the probability of moving from state i to j, and p_{ii} the probability of remaining in state i if the previous state was also i.

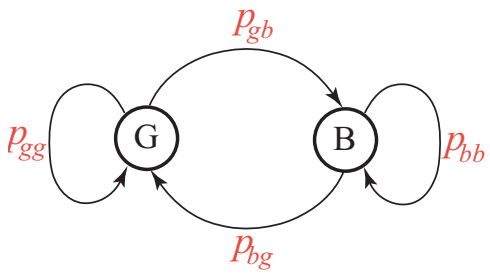

Figure 4.6: *Gilbert-Elliott channel model.*

From basic probability theory:

$$p_{gb} + p_{gg} = 1 \tag{4.7}$$

$$p_{bb} + p_{bg} = 1 \tag{4.8}$$

The Gilbert-Elliott Model is a first order, 2-state Hidden Markov Model, thus we can write the transition matrix, M, as:

$$M = \begin{bmatrix} p_{gg} & p_{gb} \\ p_{bg} & p_{bb} \end{bmatrix} \tag{4.9}$$

which, by using Eqs. (4.7)(4.8), can be rewritten as:

$$M = \begin{bmatrix} p_{gg} & p_{gb} \\ p_{bg} & p_{bb} \end{bmatrix} = \begin{bmatrix} 1-p_{gb} & p_{gb} \\ p_{bg} & 1-p_{bg} \end{bmatrix} \tag{4.10}$$

However, before proceeding, we have to find the relation between the variables of our channel (average burst length, $\bar{l}$, and probability of packet loss, p_{pl}) and the variables of the Gilbert-Elliott model (p_{bg} and p_{gb}).

The first relationship is easy to calculate and can be shown to be:

$$\bar{l} = \frac{1}{p_{bg}} \tag{4.11}$$

Calculating the second relationship is a little bit more laborious. Let us start by writing the local balance equations along with the property that the sum of all state probabilities has to be one:

$$p_{gb}\pi_G - p_{bg}\pi_B = 0 \tag{4.12}$$

$$\pi_G + \pi_B = 1 \tag{4.13}$$

It is immediate to verify that:

$$p_{gb} = \frac{p_{bg}\pi_B}{1-\pi_B} \tag{4.14}$$

Finally, by making use of eqs. (4.11) and (4.14) our model remains totally determined by the parameters of the real channel:

$$\begin{aligned} p_{bg} &= \frac{1}{\bar{l}} \\ p_{gb} &= \frac{\pi_B}{\bar{l}\,(1-\pi_B)} \end{aligned}$$

Where we would like to highlight that the probability of packet loss is the probability of being in the *bad* state, $p_{pl} = \pi_B$.

First of all, we should check if the simulated channel effectively corresponds to the one described by the design parameters. Thus, we ran the simulations several times under different channel parameters and analyzed the resulting systems. The output of these analysis are shown in Table 4.4.

Desired Channel		Simulated Channel	
π_B	$\bar{l}$	π_B	$\bar{l}$
0.1	11	0.099	10.643
0.2	5	0.199	4.885
0.3	8	0.302	8.089

Table 4.4: *Testing the channel.*

The main reason for introducing the channel was to check the robustness to packet losses of the compression algorithm. In Fig. 4.7 we can graphically see the destructive effect of the bursts of errors. There are two bursts spreading over the time ranges $[474 - 493]$ and $[503 - 518]$ where the reception of packets is interrupted (it can be deduced that we are dealing with bursts from the fact that the decoded data remain constant, not being able to follow the evolution of the real data). As it was programmed, we stick to the last correctly received temperature measure, committing decoding errors during these ranges. To study the performance of the algorithm in terms of probability of decoding error, a set of simulations were carried out, the results being reported in Table 4.5.

	Average Burst Length				
Packet Loss Rate	1	3	5	10	15
0 %	0	0	0	0	0
10 %	$7.33 \cdot 10^{-2}$	$7.5 \cdot 10^{-2}$	$7.83 \cdot 10^{-2}$	$8.12 \cdot 10^{-2}$	$9.45 \cdot 10^{-2}$
20 %	$1.56 \cdot 10^{-1}$	$1.58 \cdot 10^{-1}$	$1.81 \cdot 10^{-1}$	$1.85 \cdot 10^{-1}$	$1.95 \cdot 10^{-1}$
30 %	$2.42 \cdot 10^{-1}$	$2.71 \cdot 10^{-1}$	$2.74 \cdot 10^{-1}$	$2.81 \cdot 10^{-1}$	$2.87 \cdot 10^{-1}$

Table 4.5: *Probability of decoding error for several experimental setups.*

Prior to drawing any conclusions, it should be noticed that we are superimposing the probability of packet losses to that of making a decoding error due to the prediction and decoding algorithm (recall that this probability was set to 0.01). Having this in mind, there are three major results to be highlighted from Table 4.5:

a) The first one is to notice that for a probability of having zero packet

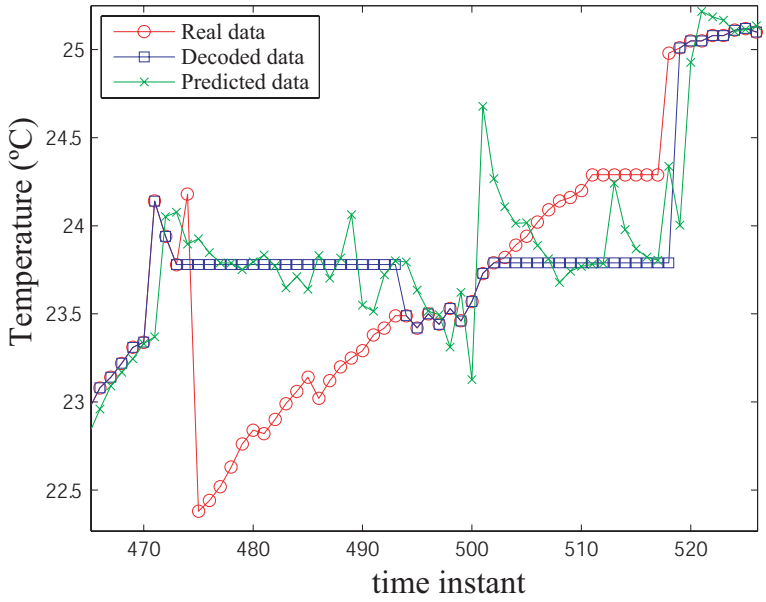

Figure 4.7: *Effect of the burst noisy channel.*

losses, the algorithm is able to perform without doing any errors;

b) The second thing to highlight is that the longer the bursts, the larger is the probability of decoding errors;

c) The third thing to remark is that for every simulation carried out, the achieved probability of decoding error was smaller that the actual packet losses;

From the previous observations we can conclude that the algorithm is robust to packet loss.

Decoding error Anytime the decoded measure does not match the real measure we say to have a decoding error. Recall we set the probability of decoding error to be $P_e = 0.01$, and that it was a basic parameter for choosing the number of bits we wanted to receive from the sensors (1.18) along with the use of Chebyshev's bound. It was also stated that Chebyshev's inequality was a too loose theoretic bound. The purpose of this paragraph is to experimentally motivate this statement. In this sense, a comparison between the tolerable noise and the prediction noise was carried out. By tolerable noise we mean the amount of noise that can exist between the prediction of a sensor

reading and the actual sensor reading without inducing a decoding error, and is calculated as $2^{i-1}\Delta$, where i is the number of requested bits and Δ is the quantization step. On the other hand, with prediction noise we just denote the difference between the value of the estimate and the actual sensor reading. A plot of the tolerable prediction noise versus the actual prediction noise is given in Fig. 4.8.

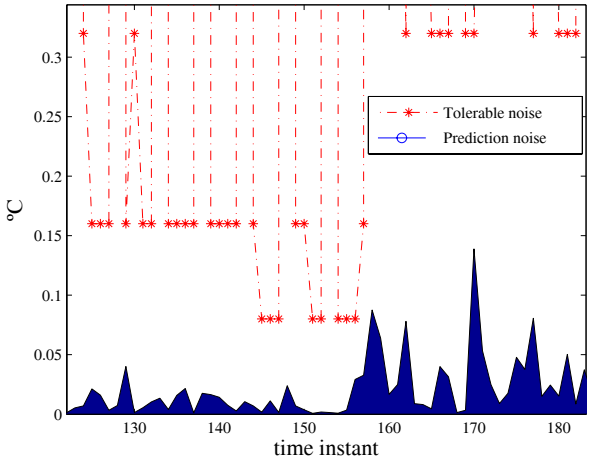

Figure 4.8: *Tolerable versus prediction noise.*

From Fig. 4.8 we can conclude that the tolerable noise is much higher than the actual prediction noise. This is because we chose a non-aggressive policy when determining the number of necessary bits to request data to the sensors. If we choose a more aggressive policy, we will be able to achieve an improved compression gain, but we will also get closer to the probability of error we set. For example, for a $P_e = 0.01$ we propose to use the following heuristic formula to calculate the number of bits, instead of using (1.18).

$$i = \frac{1}{2}\log_2\left(\frac{\sigma^2_{N_j}}{\Delta^2 P_e}\right) + 0.1 \tag{4.15}$$

If we simulate once again with this new formula, we obtain a compression gain of a 42% and a probability of error equals $9.67 \cdot 10^{-3}$, what clearly outperforms the results drawn when we used (1.18). If

we plot again (Fig. 4.9) the tolerable noise versus the prediction noise we can see how our margin has been reduced below the quantization step.

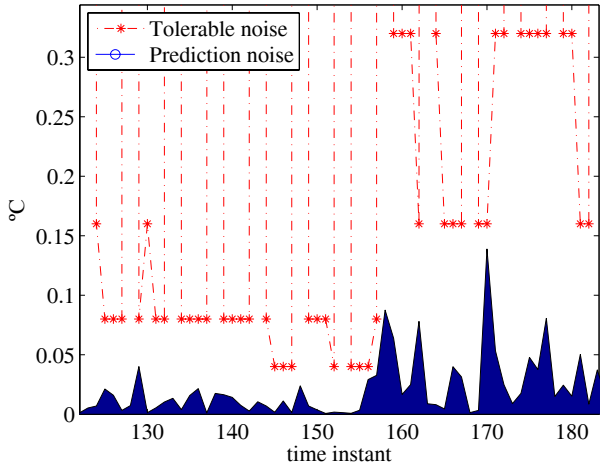

Figure 4.9: *Tolerable versus prediction noise with improved number of requested bits, i.*