Performances of Multi-Hops Image Transmissions on IEEE 802.15.4 Wireless Sensor Networks for Surveillance Applications

Congduc Pham
University of Pau, LIUPPA Laboratory
congduc.pham@univ-pau.fr

Vincent Lecuire
CRAN Laboratory
vincent.lecuire@cran.uhp-nancy.fr

Jean-Marie Moureaux
CRAN Laboratory
jean-marie.moureaux@cran.uhp-nancy.fr

***Abstract*—Surveillance applications with Wireless Sensor Networks can be strengthened by introducing imaging capability: intrusion detection, situation awareness, search&rescue... As images are usually bigger than scalar data, and a single image needs to be split in many small packets, image transmission is a real challenge for these applications, especially when knowing that the wireless medium in WSN has high throughput limitations and high packet loss rates due to numerous wireless channel errors and contention. Our contribution in this paper is on identifying limitations and bottlenecks of sensor board hardware and 802.15.4 radio to determine the performance level that can be expected when transmitting still images on a multi-hop network. In this paper, we will present experimentations with real sensor boards and radio modules. We will highlight the main sources of delays assuming no flow control nor congestion control to determine the best case performance level. The objective here is to present the potentials and the limitations of image-based wireless sensor networks.**

***Index Terms*—Sensor networks, robust image transmission, video surveillance, mission-critical applications**

I. Introduction

In the last few years, the research efforts in the field of Wireless Sensor Networks (WSN) were focused more particularly on low cost scalar sensor nodes. This type of networks composed of a large variety of nodes is able to gather on a large scale environmental data such as temperature, acceleration, moisture level, etc, and to carry out specific processing (fusion, compression,...) on these data to transmit obtained information to a base station (Sink) for further treatment and analysis. However, the purely scalar nature of these collected data might be limiting for more complex applications such as object detection, surveillance, recognition, localization, and tracking. Therefore, in addition to traditional sensing network infrastructures, a wide range of emerging wireless sensor network applications can be strengthened by introducing a visioning capability. The vision capability is a more effective means to capture important quantity of richer information and vision constitutes a dominating channel by which people perceive the world. Nowadays, such applications are possible since low-power sensors equipped with a visioning component already exist. This article therefore considers Wireless Image Sensor Networks (WISN) where sensor nodes are equipped with miniaturized visual cameras to provide visual information. These image sensors can be thrown in mass on an area of interest for search&rescue situation awareness or intrusion detection.

There have been studies on image/multimedia sensors [1], [2], [3], [4], [5] but few of them really consider timing on realistic hardware constraints for sending/receiving packets. [5] is probably the closest work to ours with real experimentations on iMote2 sensors. However, their focus was more on global performances than on a detailed study of the hardware and API limitations. In this paper, we will present experimentations with real sensor boards and real radio modules to transmit still image encoded with an optimized approach that offers both small image size and robustness. We will highlight the main sources of delays assuming no flow control nor congestion control to determine the best case performance level. One usage for this study could be to use these real performance measures in simulation models to provide more realistic performances for large-scale image sensor networks deployment for instance. Although it is not possible to address the large variety of existing sensor boards (see [6] for a quite exhaustive list of existing sensor boards) we however provide measures for UART-based and SPI-based sensors that could be adapted to other type of sensors to determine the performance level that can be expected. The motivation of this article is to present the potentials and the limitations of image-based wireless sensor networks for surveillance applications when using 802.15.4 multi-hop connectivity.

The paper is then organized as follows: Section II presents real measures on sensor hardware and radio modules of what could typically be expected with 802.15.4 communication stacks at the application level. Section III presents an image encoding scheme that offers both small image size and robustness suitable for image transmission over low bandwidth and lossy channels. Experimental results of multi-hop still image transmissions with 802.15.4 radio modules will be presented in Section IV. Conclusions will be given in Section V.

II. Communication performances on real sensors

A. Sending performances

One of the main objectives of our work in this paper is to take into account the real overheads and limitations of realistic sensor hardware. Most of simulation models or analytical studies only consider the frame transmission time as a source

of delay. However, before being able to transmit a frame, the radio module needs to receive the frame in its transmission buffer. In many low cost sensor platforms, the bottleneck is often the interconnection between the microcontroller and the radio module. Many sensor boards use UARTs (serial line) for data transfer with communication rates between 38400bps and 230400bps for standard baud rates. Non-standard baud rates are usually possible, depending on the microcontroller master clock, and also, depending on UARTs, higher speed can be achieved. Nevertheless, in addition to the radio transmission time, one has to take into account the time needed to write data into the radio module's buffer. This time is far from being negligible as most of serial communications also adds 2 bits of overhead (1 start bit and 1 stop bit) to each 8-bit data. Therefore, with a serial data transfer rate 230400bps, which is already fast for a sensor board UART, writing 100 bytes of application payload needs at least $100 \times 10/230400 = 4.34ms$ if the 100 bytes can be passed to the radio without any additional framing bytes. In many cases, one has to add extra framing bytes, making the 4.34ms a sort of minimum overhead to add to each packet transmission in most of UART-based sensor boards. If we consider an image transmission that requires sending the image in many packets, we clearly see that the minimum time before 2 packet generation is the sum of the time to write frame data to the radio and the time for the radio to transmit the frame. According to the 802.15.4 standard, if we consider a unicast transmission with the initial back-off exponent BE set to 0 (default is 3), we typically need a minimum of $5.44ms + 4.34ms = 9.78ms$ to send a single 100-byte packet if there is no error. Now, in high-end sensor boards such as the iMote2 from Crossbow released a few years ago, the radio module is connected to the microcontroller through a high-speed bus (SPI for instance) which allows for much higher data transfer rates, in which case a unicast transmission of a single 100-byte packet with the same MAC parameter would take $5.44ms + \epsilon$. However, as we will how later on, not only the sending side should be taken into account and sending fast is usually not reliable.

To highlight the importance of the time needed to write to the radio on some hardware, we measure on real sensor hardware and communication API the time spent in a generic send() function (most communication APIs have a function to send a packet), noted t_{send}, and the minimum time between 2 packet generation, noted t_{pkt}. t_{pkt} will typically take into account various counter updates and data manipulation so depending on the amount of processing required to get and prepare the data, t_{pkt} can be quite greater than t_{send}. With t_{send}, we can easily derive the maximum sending throughput that can be achieved if packets could be sent back-to-back, and with t_{pkt} we can have a more realistic sending throughput. In order to measure these 2 values, we will use a traffic generator that sends packet back-to-back with a minimum of data manipulation needed to maintain some statistics (counters) and to fill-in data into packets. When possible, we also add non-intrusive accurate timing of the programming API.

We will consider Libelium WaspMote [7] that are used in a number of Smart Cities and environmental monitoring projects [8], [9], Arduino MEGA 2560 [10] (Libelium WaspMote IDE is largely based on Arduino) and the iMote2 mote [11] which is a powerful evolution of Mica2 and MicaZ motes, both platforms being well-known to the WSN research community.

1) Libelium WaspMote & Arduino: Libelium WaspMote use an IEEE 802.15.4 compliant radio module called XBee manufactured by Digi [12] which offers a maximum application level payload of 100 bytes. By default, the XBee module uses a $macMinBE$ value of 0 while the default value for IEEE 802.15.4 is 3. The WaspMote features an Atmega1281 running at 8MHz. The XBee module and the micro controller communicate through an UART, and for the WaspMote the default data rate is set to 38400bps by the Libelium API. In a first step we will investigate the off-the-shelves performance of the WaspMote. However we use a modified version of the "light" Libelium API that provides much higher performance level compared to the "full" Libelium API that additionally handles long packets with fragmentation/reassembly support. As WaspMote is very similar to the well-known Arduino boards (WaspMote IDE is actually based on the Arduino IDE) the results presented in this section also apply to the Arduino MEGA 2560 board which features an ATmega2560 running at 16MHz. This Arduino board is one of the fastest Arduino boards in the market and is quite representative of UART-based sensor boards. On the Arduino, we also use a very lightweight communication library[13] and our modified API for the WaspMote achieves the same level of performance than the Arduino's API.

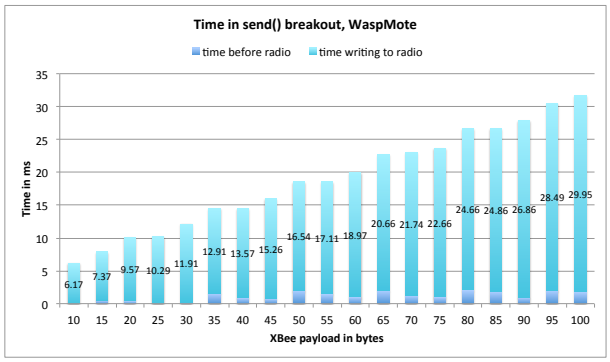


Fig. 1. Time in send() breakout, WaspMote

Figure 1 shows the time in send() breakout for the WaspMote (data transfer rate is 38400) where we can especially see the time required to write to the radio. These results are averaged over 20 packets for each packet size and even if the tests can not be totally deterministic the statistical variations are small enough. The sum of all the timing represents what we called t_{send}. The "time before radio" is the time to prepare data before writing to the radio. We can see that the main bottleneck here is the time to write to the radio as the data transfer rate is only 38400bps.

Figure 2 shows both t_{send} and t_{pkt} for the WaspMote. The maximum realistic throughput could be derived from t_{pkt}. On

the Arduino, the communication API is a bit more efficient and t_{send} is roughly the time to write to radio: for a 100-byte packet t_{send} (averaged) is found to be 29.8ms and t_{pkt} is 33.45ms.

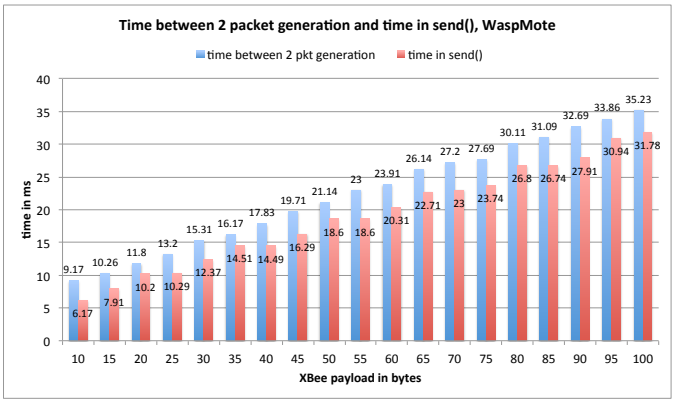


Fig. 2. Time between 2 packet generation and time in send(), WaspMote

We then increased the UART data transfer rate that is set by default to 38400bps. However, increasing the baud rate cannot be done without taking into account some timing constraints that may make the serial communication unreliable [14]. The WaspMote microcontroller runs at 8MHz while the XBee module has an 16MHz clock and requires that the frequency is 16 times the baud rate. It means that for a baud rate of 38400, the actual operating frequency need to be $16 \times 38000 = 614400$Hz. For reliable communication, the WaspMote clock should also produce a frequency close to 614000Hz. Since it runs at 8MHz, the dividing factor is $8000000/614000 = 13.020833$. Using the nearest integer dividing factor of 13, the actual baud rate is $8000000/16/13 = 38461.54$ which is 1.0016026 times greater than the target baud rate. The error is about 0.1602% which allows for reliable communication between the microcontroller and the XBee module. Actually, 38400, which is the value chosen by the Libelium API is the fastest standard baud rate that provides acceptable errors between the target baud rate and the actual baud rate. Using 57600 or 115200 baud rates would generate too many errors, making the communication very unreliable and therefore not functioning at all. Even on the XBee, 57600 and 115200 baud rates can not accurately be achieved with the 16MHz clock. Using these constraints, the perfect dividing factors for the WaspMote are 10, 5, 4, 2 and 1 which correspond to 50000, 100000, 125000, 250000 and 500000 baud rates respectively. As the maximum IEEE 802.15.4 effective throughput is roughly 166666bps in broadcast mode with no errors, there is no point to consider 500000 baud rate that would additionally overflow the transmission buffer. On the Arduino, as the clock runs at 16MHz, there is no problem in getting these baud rates with a dividing factor of 20, 10, 8, 4 and 2 respectively.

With a serial transmission of 8 data bits, 1 start bit and 1 stop bit that gives 10 bits per byte, Figure 3 shows the estimated time to write to radio if baud rates up to 250000 were applied. If we assume that reducing the time to write to radio does not change the other overheads, we can estimate the new t^B_{send} for a baud rate B higher than 38400 as follows:

$$t^B_{send} = t^{38400}_{send} - timeToWriteToRadio^{38400} + timeToWriteToRadio^B$$

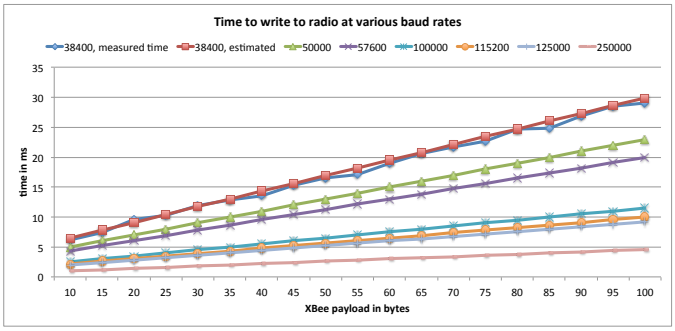


Fig. 3. Time to write to radio at various baud rates

Then, assuming that the overheads between 2 packet generation are also independent from the time to write to radio, we can estimate the new t^B_{pkt} for a baud rate B higher than 38400 as follows:

$$t^B_{pkt} = t^{38400}_{pkt} - t^{38400}_{send} + t^B_{send}$$

To verify our assumption and to prepare for the experimental image transmission tests using maximum achievable performance, we set the baud rate of the XBee module to 125000 and 250000 and ran again the traffic generator on the WaspMote after having changed the default data transfer rate of the Libelium communication API from the default 38400 to 125000 and 250000. Figure 4 shows the estimated and measured time between 2 packet generation for data transfer rates of 125000 and 250000 with the WaspMote.

We can see that the estimated and the measured curves are very close each other, thus validating our estimation method of the time to write to radio and the constant overheads of the communication API. We did the same for Arduino and in summary, when using a 100-byte payload, we can have $t^{125000}_{pkt} \approx 16ms$ and $t^{250000}_{pkt} \approx 12ms$ for the WaspMote; and $t^{125000}_{pkt} \approx 13ms$ and $t^{250000}_{pkt} \approx 7ms$ for the Arduino. Note that the send() function may returns before the entire packet has been transmitted or before the transmission status has been received, thus explaining the 7ms that is measured while the minimum transmission time was found previously about 10ms for a 100-byte packet. The results for the Arduino platform can be seen as the minimum time between 2 packet generation that could be achieved for sensor nodes with a similar architecture using UART lines for communications between microcontroller and radio module. However, we observed many packet losses at 250000 baud rate which was not the case at 125000 baud rate. Therefore, for maximum reliability, from now on, we will use the WaspMote and the Arduino with a serial data transfer rate of 125000bps. We will see in the next section that we will have to take into account the limitations at the receiver side.

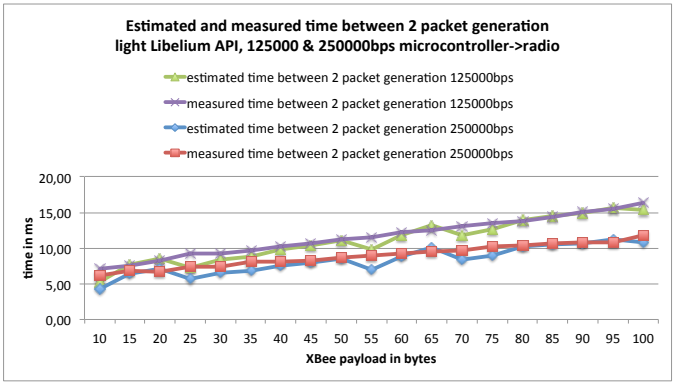
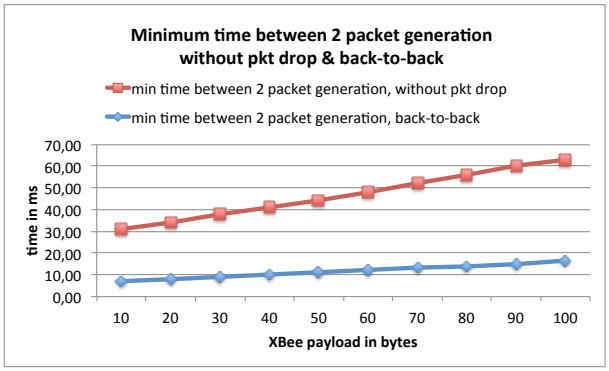


Fig. 4. Estimated and measured for t_{pkt}^{125000} and t_{pkt}^{250000}



0

Fig. 5. Minimum time between 2 packet generation with a receiver

2) Crossbow iMote2: For the iMote2, we use the .NET version with the MicroFramework environment. The iMote2 is a high-end board with a TI CC2420 802.15.4 radio module. As opposed to WaspMote and Arduino, communication between the microcontroller and the CC2420 is realized through an SPI bus which data transfer rate is much higher than UART serial communication. Once again, with a traffic generator, we measured the time in send() and found it very small: 2 to 3ms for a 100-byte payload. Therefore, on the iMote2 the microcontroller-radio interconnection is clearly not the bottleneck. However, as the 802.15.4 transmission time of a frame of maximum size takes roughly 5ms and as the CC2420 radio has only room for a full-size frame (128 bytes of buffer), the time between 2 packet generation should at least be set greater than this value.

B. Receiver performances

In the next set of experiments, we use the traffic generator to send packets to a receiver. In general, flow control and congestion control can be implemented but any control would slow down the transmission anyway. Therefore, we are not using flow control nor congestion control but experimentally determine the minimum time between 2 packet generation at the sending side that would not overflow the receiver.

Figure 5 shows for the WaspMote the minimum time between 2 packet generation to avoid frame drops or incomplete frames at the receiver. We can see that with a receiver and the concern that packets are not arriving too fast at the receiver side, the minimum time between 2 packet generation increases from $\approx 16ms$ to $\approx 63ms$ for the WaspMote for the maximum payload size! On the Arduino, we do have the same behavior: from $\approx 12ms$ to $\approx 40ms$. For the iMote2, while the sending time can be very small, it appeared that due to the very small radio buffer size in the TI CC2420, the time between 2 packet generation need to be increased to $\approx 80ms$ to keep the packet losses rates below 10% at the receiver! This is a result that we did not expect and that was quite surprising. We observed the same problem on a SoftBaugh board built around an MSP430 microcontroller with the CC2420 radio module. Although this is not a result that could be considered true for other high-end boards, it appeared here that the iMote2 board under .NET framework can not sustain high speed data reception rates.

C. Multi-hop issues

In a WISN, images are sent from sensor nodes to a sink or base station. This sink is not always the final destination because it can also transmit the images to a remote control center, but it is generally assumed that the sink has high bandwidth transmission capabilities. Figure 6 shows a detailed time diagram of a multi-hop transmission. We can see that all the sensor nodes along the path from the source node to the sink do have the same constraints regarding the minimum time between 2 packet generation.

Actually, it is well-known that multi-hop transmissions generate higher level of packet losses because of interference and contention on the radio channel (uplink, from the source; and downlink, to the sink). In this case, when the minimum time between 2 packet generation is too small, there are contention issues between receiving from the source and relaying to the sink. This is depicted in figure 6 by the gray block.

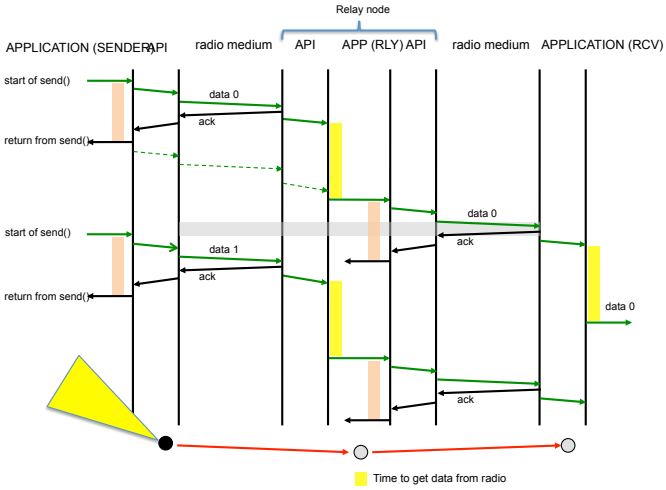


Fig. 6. Multi-hop transmission of images

However, as we found that the minimum time between 2 packet generation is much greater than the radio transmission time (about 5ms for a 100-byte packet), multi-hop transmissions in this case will most likely rather suffer from high processing latencies than from contention problem. On the

figure, we can see that the relay node, upon reception of the packet from the source node, needs an additional delay to get data from the radio (yellow block), before being able to send it to the next hop. This delay is far from being negligible as in the best case it is similar to the time to write to the radio.

For the WaspMote and the Arduino boards, we also found that the time needed to read the received data, noted t_{read}, is quite independent from the communication baud rate between the microcontroller and the radio module. In all our experimentations, for baud rates of 38400, 125000 and 250000, t_{read} remains constant and depends only on the data size. Figure 7 plots t_{read} for both the WaspMote and the Arduino. The reason why t_{read} only depends on the data size, at least at the application level, is as follows: most of communication API used a system-level receive buffer and when a packet arrives at the radio, a hardware interrupt in raised and appropriate callback functions are used to fill in the receive buffer that will be read later on by the application. Therefore, the baud rate has only an impact on the time needed to transfer data from the radio module to the receive buffer. When in the receive buffer, the time needed to transfer the data from the receive buffer to the application depends on the speed of memory copy operations, therefore depending mainly on the frequency used to operate the sensor board and the data bus speed. As we can see in figure 7, t_{read}, for a 100-byte packet, is about 50ms and 35ms on the WaspMote and Arduino respectively.

In total, when adding additional data handling overheads, a relay node based on a WaspMote needs about 108ms to process the incoming packet and to relay it to the next hop, once again for a 100-byte packet. The Arduino can do it in about 94ms, see Figure 7, blue and green curves.

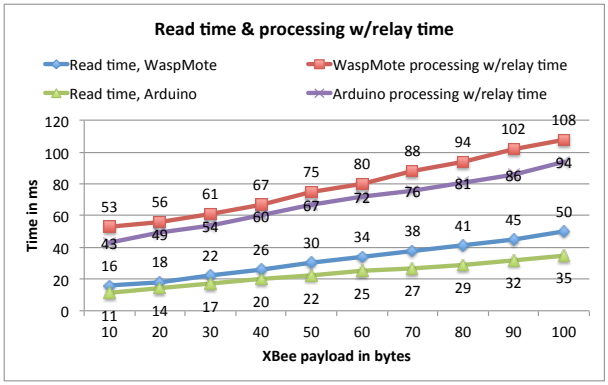


Fig. 7. Measured time to read data

In case the next packet from the source node arrives before the previous packet has been read, the reception buffer may overflow quite quickly. This case is depicted by the dashed green arrow from the source to the first relay node. On more elaborated OS and processors, it is possible to have a multi-threaded behavior to process earlier the received packet but in this case contention on serial or data buses need to be taken into account. In all cases, we clearly see that in the best case the next packet will not be sent before the return of the last send. We can see that multi-hop transmission on this type of platform adds a considerable overhead that put strong constraints on the image encoding scheme.

On the iMote2 we measured t_{read} and found it very similar to the time in send() measured previously at about 2 to 3ms. As the iMote2 radio module is connected with an SPI bus this result is quite consistent. The iMote2 can theoretically receive and relay a 100-byte packet in about 15ms but as the reception rate was found previously to be very limited, multi-hop communication with iMote2 has the same level of performance than with Arduino nodes.

In summary, we found that the WaspMote, the Arduino and the iMote2 boards have all strong limitations, mostly in receiving packets where the overhead of memory and data manipulation puts a minimum inter-packet time at the sending side to roughly between 90ms and 110ms. In this context, even an 128x128 image in raw format (16384 bytes) would require between 14s and 18s to be transmitted in the best case, assuming 100-byte packets. Table I summarizes the main communication delays on the sensor motes we study in this paper. WaspMote and Arduino use 125000 baud rate.

Sensor platform	WaspMote	Arduino	iMote2
t_{send} (ms)	11	10	3
t_{pkt} (ms)	16	13	6
t_{rcv} (ms)	63	40	80
t_{read} (ms)	50	35	3
t_{rly} (ms)	108	94	91

TABLE I
SUMMARY OF TIME CONSTRAINTS

In the next section, we will present an optimized image coding scheme to efficiently reduce the image size and increase the tolerance to packet losses and reception order.

III. Optimized image coding scheme

Transmitting images on WSN faces numerous constraints such as transmission latency, energy efficiency and reliability. For surveillance applications, it is not very tractable to increase reliability with retransmission mechanisms, and usage of redundant information for error correction is very costly. Early studies have confirmed that image communication needs to be especially tolerant to packet losses [15] which automatically make traditional JPEG compression scheme unsuitable as it suffers from very high spatial correlation: an entire image could be impossible to decode with only a few packets missing. We therefore propose an optimized encoding scheme based on the 2 following key points:

1) Image compression must be carried out by independent block coding in order to ensure that data packets correctly received at the sink are always decodable.
2) De-correlation of neighboring blocks must be performed prior to packet transmission by appropriate interleaving methods in order to ensure that error concealment algorithms can be efficiently processed on the received data.

The first point motivated the choice of block-based compression schemes. Among them, we chose a DCT-based method for

sake of simplicity implementation in the context of WSN. In the following sections we detail first the proposed compression scheme and, second, the proposed interleaving method.

A. Proposed compression scheme

Our image compression scheme is referred to as a JPEG-like coder. It is based on the well known Discrete Cosine Transform (DCT)-Scalar Quantization (SQ)-Entropy coding chain applied on 8x8 pixel blocks. JPEG has shown its efficiency in terms of rate-distortion tradeoff for real-life image compression for many years, especially at moderate compression ratios, which are of interest for WSN applications. However, the computational cost of JPEG is high regarding the resource limitations of sensor nodes in terms of processor speed and thus may contribute to increase the energy consumption while it is used to reduce it. Thus, it is crucial to minimize the number of operations required to encode pixel blocks, in particular at the DCT step. The Arai-Agui-Nakajima DCT (AAN) algorithm [16] is known to be the most efficient 1D DCT for DCT coefficients that need to be quantized, which is the case in the JPEG-like coders. It requires only 5 multiplications and 29 additions for an 8 point DCT, which leads to 80 multiplications and 464 additions for 2D blocks. Furthermore, SQ step requires 64 multiplications per block. To still reduce the processing time, it is important to use fixed-point arithmetic for calculation. At the end of the chain, binary encoding operations could be also reduced by using jointly Golomb and Multiple Quantization coders instead of Huffmann coding [17] which leads to an optimized compression ratio/energy consumption trade-off, with respect to the application. More details on tuning of compression parameters, in particular the quality factor (Q), of the enhanced encoding scheme is detailed in [18].

B. Proposed block interleaving scheme

As stated previously, packet losses are inherent to WSN, thus, packetization of JPEG encoded data should account it, in order to allow error concealment methods to be efficient at the decoder. To achieve this, each packet sent by the camera node must contain an integer number of encoded blocks, so that, at the decoder side each block is entirely received or entirely lost. However, it is obvious that in case of loss of several neighbor blocks, spatial error concealment performance decreases significantly. Thus, combining a block interleaving method with JPEG-like encoding should improve visual quality of reconstructed images as the probability that packet loss affects adjacent blocks of pixel decreases significantly. Here we propose to use the method designed in [19] which has shown both its efficiently in terms of visual quality after reconstruction and computational complexity. Indeed, only four multiplications, two additions and two divisions modulo per block are required.

Finally, the combination of the fast JPEG-like proposed encoder with the proposed block interleaving method allows to an efficient tuning of the compression ratio/energy consumption trade-off while maintaining an acceptable visual quality in case of packet loss. Thus, it participates to the improvement of the lifetime of the entire WSN. Figure 8 and Figure 9 show a 128x128 and 200x200 image respectively with various quality factor (Q). We set the maximum image payload per packet to 90 bytes because some bytes need to be reserved in the 802.15.4 payload for storing image information such as the offset in the image of each data packet received.

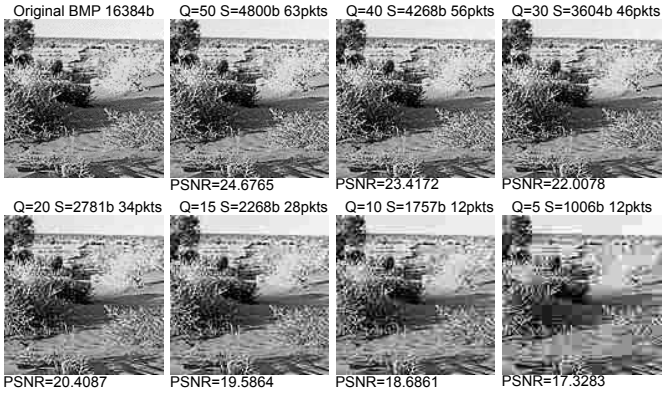


Fig. 8. 128x128 image, typical of intrusion detection applications

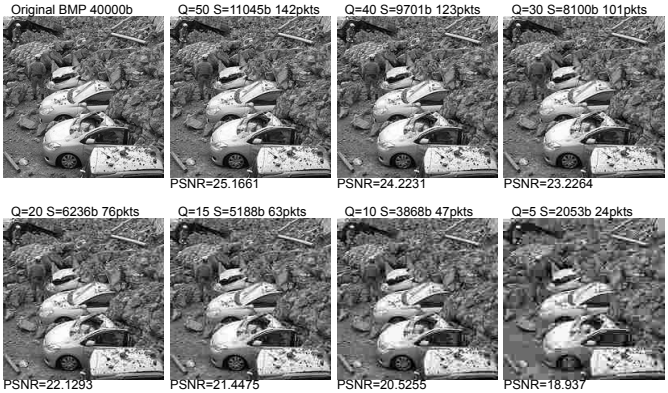


Fig. 9. 200x200 image, typical of search&rescue applications

The number of generated packets, the total size of the compressed image and the PSNR compared to the original image are shown in the figure. We can see that a quality factor of 20 is visually still acceptable while providing a good compression ratio: 5.89 for the 128x128 image and 6.14 for the 200x200 image. This is the value for Q that we will choose for the next experiments but one could imagine setting Q according to network congestion level for instance. The 128x128 size can typically be used for critical surveillance applications, such as intrusion detection that need the lowest latency, while the 200x200 size can be used for a situation-awareness application which is less delay constrained.

IV. Performance of still image transmissions

A. Experimental test-bed and results

The experiment uses 1 source node consisting of an Arduino Mega2560 with an XBee module connected to the micro

controller at 125000bps. The images are stored on an SD card and we can dynamically select which file is going to be sent, see Figure 10(left). The image file is fragmented in a number of packets according to the encoding scheme. We choose a quality factor of 20 that presents an acceptable quality for a small image size, see section III. When the sending is triggered, we can choose the time between 2 packet generation.

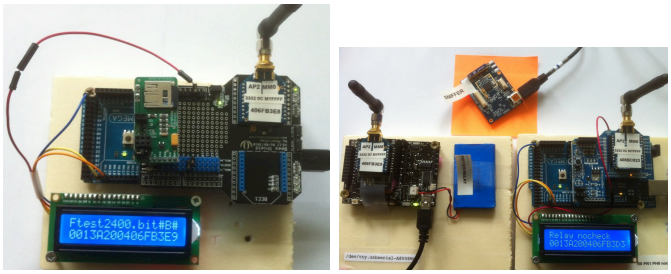


Fig. 10. Left: Arduino Mega2560 for sending images stored in the SD card. Right: relay nodes, from left to right: Libelium WaspMote, iMote2 and Arduino Mega2560

We then have 3 different relay nodes: 1 Libelium WaspMote and 1 Arduino Mega 2560 with an XBee module (once again the XBee module is connected to the micro controller at 125000bps), and 1 iMote2 with a CC2420 radio. These relay nodes are programmed to relay incoming packets to the sink which is, in our case, an XBee module connected to a Linux computer running the reception program to receive the image packets and display the image. Figure 10(right) shows the 3 types of relay nodes.

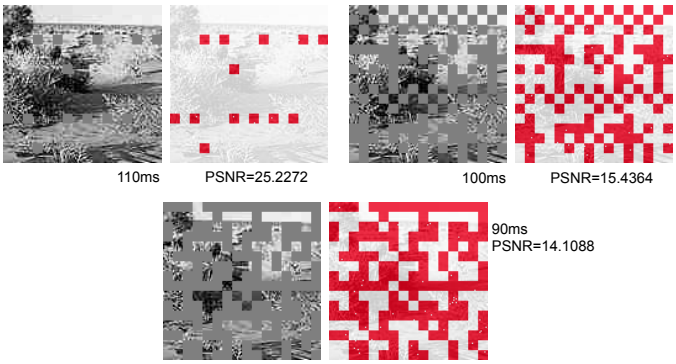


Fig. 11. 128x128 image received, Libelium WaspMote relay

Figure 11 shows for the 128x128 desert image the various reception qualities when the time between 2 image packets is varied. These results are for the Libelium WaspMote relay node. For inter-packet time greater than 110ms, the image is received without any packet loss. At 110ms, the time needed to send the image is about 3.86s. In all cases, the relay node needs about 102 to 111ms to process an incoming packet and to relay it to the next hop. This time, noted T_R can be considered as the minimum relay time introduced at every WaspMote relay node. The image reception latency at the sink with 1 relay node is about $3.86 + T_R$ assuming that the reception from serial port on a computer negligible.

Figure 12 shows for the 200x200 rescue image the various reception qualities when the time between 2 image packets is varied. The relay node is an Arduino.

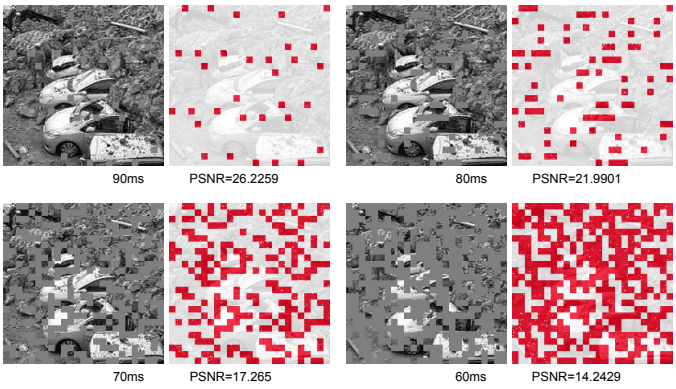


Fig. 12. 200x200 image received, Arduino relay

For an inter-packet time greater than 90ms, the image is received without any packet loss. The time for sending the image with 90ms inter-packet time is about 7s. This time can decrease to about 6s when using 80ms between each packet at the cost of more packet drops. The Arduino relay node needs about 92ms to 100ms to process an incoming packet and to relay it to the next hop. With an iMote2 as relay node, due to the limited reception rate, we have similar results than for the Arduino relay case.

B. Discussions

On the Arduino Mega2560 board using 90ms for the inter-packet time gives low packet drop rates and the 128x128 desert image with a quality factor of 20 can be sent in about 3.18s. This is probably tractable for critical surveillance applications such as intrusion detection. One can decrease the quality factor or send faster at the cost of packet drops to have much smaller transmission time: with Q=10 and 80ms inter-packet time, the image can be sent in a bit more than 1s. For situation-awareness applications, a single 200x200 image can be obtained in about 7s with Q=20. Assuming a optimized scheduling strategy for the image nodes we can assume that the control center can get 100 images from 100 different locations in about 12 minutes. Although these results could be improved by a number of optimizations to grab a few ms on the same hardware platforms, we believe that these values are quite typical of what can be found on existing sensor boards and radio modules. At least, it is much more realistic than assuming only the cost of radio transmission as it is usually the case in many simulation studies.

One source of high improvement can however come from the relay node: we use in our experiments an application-level relaying feature. It means that the packet must go up to the application level to then be re-transmitted. If the relaying is performed at the MAC layer as some radio firmware allows it, such as the DigiMesh firmware available on the XBee module, the relaying time could be reduced. We could not

however tested this feature of DigiMesh because it was not possible to accurately perform specific timing measures of the MAC relaying mechanism and of the AODV-based embedded routing protocol. A more general solution could be to perform the relaying not at the MAC level but once the packet is stored in the reception buffer at the lowest level of the sensor board API or OS. In all these cases, we can save the time to fetch the data from the reception buffer to the application.

We chose to not address the overhead for capturing and compressing the image. In practice, in addition to the communication latency, each image adds a processing delay to get image data from the camera, to compress the image and to build the image packet. Our rationale for not having addressed these issues is because there is a very large range of possibilities, connection technologies (SPI, I2C, dedicated) and some specific hardware can also speedup the compression scheme. Moreover many camera boards have the possibility to perform some processing task independently from the main microcontroller. However, it has been reported for the Cyclops, which is one of the earliest camera sensor proposed in the research community, a capturing time of about 200ms for a 128x128 image. In addition, we do have an implementation prototype of our proposed compression scheme on an early Mica2 sensor which feature an ATmega128L microcontroller that has the same level of performance than the Libelium board and we measured a compression time of about 4s for an 128x128 image and about 10s for an 200x200 image. These values can certainly be greatly decreased with dedicated hardware.

V. Conclusions

Multi-hop image transmission on wireless sensor networks is a promising technique for surveillance applications. In this paper, we presented experimentations with various sensor boards and radio modules to highlight the main sources of delays assuming no flow control nor congestion control. The purpose of the study is to determine the best case performance level that could be expected when considering IEEE 802.15.4 multi-hop connectivity. We showed that there are incompressible delays due to hardware constraints and software API that limit the time between 2 successive packet send. However, we also showed that with an optimized image encoding scheme for both reduced image size and tolerance towards packet losses, it is possible to have latencies in the order of 3s for an 128x128 image that could suit the need of intrusion detection applications for instance. When increasing the image size for situation awareness applications, a latency of 7s is achievable for an 200x200 image with a quality factor of 20.

Our contribution can be used in various ways. For instance, more realistic values can be used to build more realistic simulation models. Also, by identifying the limitations and the bottleneck, more suitable control mechanisms could be studied and proposed. For instance, while flow control and congestion control are of prime importance we believe that traditional approaches based on buffer management or rate control are not efficient because they will add to much latency that is not compatible with surveillance applications. Our further works, based on the results presented in this paper, will rather propose to have scheduling mechanisms to explicitly prevent nodes from sending packets at the same time in the same area.

Acknowledgment

The authors would like to thank John Foster for his XBee cookbook and the very fruitful discussions on the XBee module.

References

[1] M. Rahimi et al., "Cyclops: In situ image sensing and interpretation in wireless sensor networks," in *ACM SenSys*, 2005.
[2] S. Hengstler and H. Aghajan, "Wisnap: A wireless image sensor network application platform," in *Proceedings of COGnitive systems with Interactive Sensors*, 2006.
[3] S. Misra, M. Reisslein, and G. Xue, "A survey of multimedia streaming in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 10, 2008.
[4] S. Soro and W. Heinzelman, "A survey of visual sensor networks," *Advances in Multimedia*, 2009.
[5] S. Paniga et al., "Experimental evaluation of a video streaming system for wireless multimedia sensor networks," in *Proceedings of the 10th IEEE/IFIP Med-Hoc-Net*, 2011.
[6] Wikipedia, "http://en.wikipedia.org/wiki/list_of_wireless_sensor_nodes," accessed 4/12/2013.
[7] Libelium, "http://www.libelium.com/," accessed 4/12/2013.
[8] ——, "www.libelium.com/top_50_iot_sensor_applications_ranking/," accessed 4/12/2013.
[9] SmartSantander, "http://www.smartsantander.eu," accessed 4/12/2013.
[10] Arduino, "http://arduino.cc/en/main/arduinoboardmega2560," accessed 4/12/2013.
[11] Crossbow, "http://bullseye.xbow.com," accessed 4/12/2013.
[12] Digi, "http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/," accessed 4/12/2013.
[13] A. Rapp, "http://code.google.com/p/xbee-arduino/," accessed 4/12/2013.
[14] J. Foster, "Xbee cookbook issue 1.4 for series 1 (freescale) with 802.15.4 firmware, www.jsjf.demon.co.uk/xbee/xbee.pdf," April 26th, 2011. Accessed 4/12/2013.
[15] C. Yeo and K. Ramchandran, "Robust distributed mutli-view video compression for wireless camera networks," in *In Proceedings of SPIE Visual Communications and Image Processing*, 2007.
[16] Y. Arai, T. Agui, and M. Nakajima, "A fast dct-sq scheme for images," *Transactions of the IEICE*, vol. E71, 1988.
[17] Q. Lu, W. Luo, J. Wang, and B. Chen, "Low-complexity and energy efficient image compression scheme for wireless sensor networks," *Computer Networks*, vol. 52, 2008.
[18] J.-M. M. V. Lecuire, L. Makkaoui, "Fast zonal dct for energy conservation in wireless image sensor networks," *Electronics Letters*, vol. 48, 2012.
[19] C. Duran-Faundez and V. Lecuire, "Error resilient image communication with chaotic pixel interleaving for wireless camera sensors," in *Proceedings of ACM Workshop on Real-World Wireless Sensor Networks*, 2008.