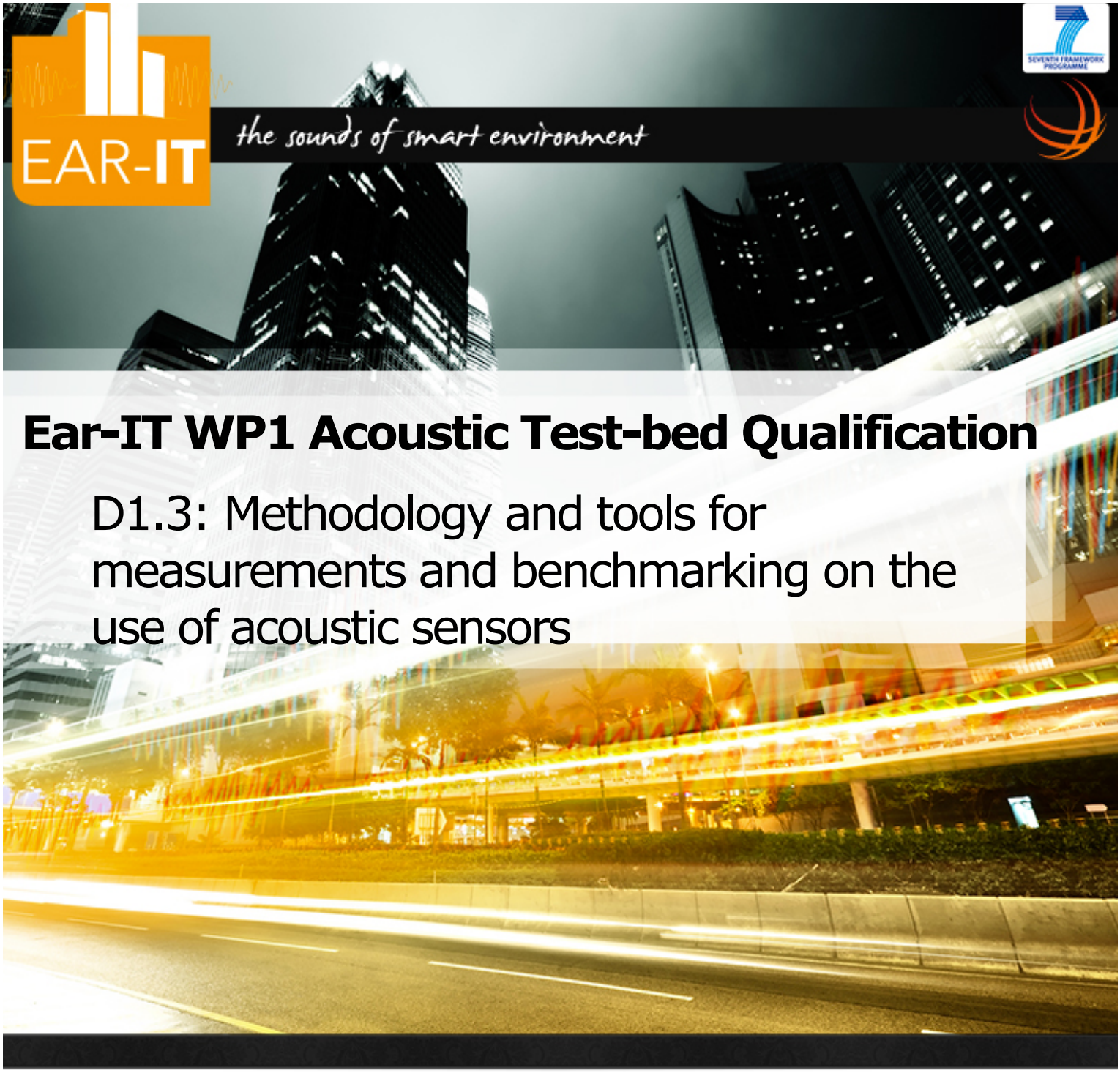


Ear-IT WP1 Acoustic Test-bed Qualification

D1.3: Methodology and tools for measurements and benchmarking on the use of acoustic sensors

Abstract

This document is the EAR-IT deliverable 1.3. It presents the methodology and tools for measurements and benchmarking on the use of acoustic sensors with a number of performance indicators. It describes the measure campaigns in Santander's SmartSantander and Geneva's HobNet test-beds to determine with the proposed methodology and tools the NETWORK performance indicators while the ENERGY indicators are measured in lab. While 1-hop transmission can be easily realized with the developed solutions, achieving high audio quality, the experimentations show that multi-hop audio quality, especially in non-LOS conditions, heavily depend on the choice of the relay nodes. However, results are promising as multi-hop audio transmission with packet loss rate below the maximum accepted threshold has successfully been tested with appropriate position of relay nodes. In addition, energy consumption has been measured and were found compatible with a smart cities environment and usage scenario.

UNINOVA Instituto de Desenvolvimento de Novas Tecnologias
Fraunhofer IDMT
UC Universidad de Cantabria
MANDAT INTERNATIONAL
CDT
SmartSensingStars

Project Number:	Project Acronym:	Project Title:
318381	EAR-IT	Experimenting Acoustics in Real environments using Innovative Test-beds

Instrument:	Thematic Priority
STREP	Future Internet Research and Experiment

Title

Methodology and tools for measurements and benchmarking on the use of acoustic sensors

Contractual Delivery Date:	Actual Delivery Date:
October 1st 2014	November 1st 2014

Start date of project:	Duration:
October, 1st 2012	24 months

Organization name of lead contractor for this deliverable:	Document version:
EGM	V1.0

Dissemination level (Project co-funded by the European Commission within the Seventh Framework Programme)		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group defined by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Authors (organizations) :

Congduc Pham, EGM
Philippe Cousin, EGM

Abstract :

This document is the EAR-IT deliverable 1.3. It presents the methodology and tools for measurements and benchmarking on the use of acoustic sensors with a number of performance indicators. It describes the measure campaigns in Santander's SmartSantander and Geneva's HobNet test-beds to determine with the proposed methodology and tools the NETWORK performance indicators while the ENERGY indicators are measured in lab. While 1-hop transmission can be easily realized with the developed solutions, achieving high audio quality, the experimentations show that multi-hop audio quality, especially in non-LOS conditions, heavily depend on the choice of the relay nodes. However, results are promising as multi-hop audio transmission with packet loss rate below the maximum accepted threshold has successfully been tested with appropriate position of relay nodes. In addition, energy consumption has been measured and were found compatible with a smart cities environment and usage scenario.

Keywords :

Acoustic data, benchmark methodology, test-bed, audio streaming

Disclaimer

Revision History

The following table describes the main changes done in the document since it was created.

Revision	Date	Description	Author (Organisation)
0.1	15th September 2014	Initial drafting	C. Pham (EGM)
0.5-0.9	1st October 2014	Development of full report and several iterations after internal review	C. Pham (EGM)
1.0	29th October 2014	Final version	C. Pham (EGM) P. Cousin (EGM)

Table of Content

1. Introduction

This document is the EAR-IT deliverable 1.3. In previous deliverable 1.2 we defined some selected performance indicators and presented the minimum requirements for use of acoustic sensors on the various EAR-IT test-beds based on WSN and IoT nodes with IEEE 802.15.4 radio technology. These performance indicators were categorized into:

1. Network performance indicators (NETWORK)
2. Audio quality indicators (AUDIO),
3. Energy indicators (ENERGY).

This document describes a benchmarking approach to provide performance indicators that would qualify the various EAR-IT test-beds based on WSN and IoT nodes with IEEE 802.15.4 radio technology. We will review the main performance issues when it comes to support acoustic data: packet loss rate, relay latency and packet jitter to name a few. We will also consider audio quality and energy aspects as part of our benchmark methodology in order to provide both performance and usability indicators.

One main motivation behind an accurate test-bed qualification on the 3 proposed indicators is the possibility of near real-time multi-hop audio streaming from a source to a control center in case of emergency, using low-resource IoT nodes (typically the legacy sensors deployed in the Santander's SmartSantander test-bed). Therefore this document will also present an overview of audio streaming techniques and the various hardware developed for this objective. The document is organized as follows:

- Chapter 2 will present:
  - a review of the EAR-IT test-beds with the associated sensor platform hardware
  - a review of the IoT node network performance obtained during the network qualification phase (see Deliverable 1.1)
  - a review of the minimum requirement for use of acoustic data (see Deliverable 1.2)
- Chapter 3 will focus on audio streaming features. We start by presenting audio streaming techniques then describe the motivation behind the developed audio board. The main characteristics of the audio board as well as the implemented services developed for enabling and demonstrating multi-hop audio streaming on low-resource IoT nodes will be presented.
- Chapter 4 will present the benchmark methodology and tools
- Chapter 5 will present our tests to determine network performance indicators
- Chapter 6 will present our tests to determine energy indicators
- Chapter 7 will describe the proposed benchmark procedure to test other test-beds
- Chapter 8 will conclude this document

2. Review of EAR-IT test-beds and developed hardware

The EAR-IT test-beds consist in (i) the SmartSantander test-bed and (ii) the HobNet test-bed. The SmartSantander test-bed is a FIRE test-bed with 3 locations. Being one location, the Santander city in north of Spain has deployed more than 5000 nodes deployed across the city. This is the site we will use when referring to the SmartSantander test-bed. HobNet is also a FIRE test-bed that focuses on Smart Buildings. Although the HobNet test-bed has several sites, within the EAR-IT project only test-bed located at MANDAT Intl and HEPIA are concerned. Many information can be found on corresponding project web site (www.smartsantander.eu and www.hobnet-project.eu) but we will present in the following paragraphs some key information that briefly present the main characteristics of the deployed nodes.

Review of SmartSantander test-bed hardware

IoT nodes and gateways

IoT nodes in the Santander test-bed are WaspMote sensor boards and gateways are Meshlium gateways, both from Libelium. Most of IoT nodes are also repeaters for multi-hops communication to the gateway. Figure 3 shows on the left part the WaspMote sensor node serving as IoT node and on the right part the gateway. The WaspMote is built around an Atmel ATmega1281 micro-controller running at 8MHz. There are 2 UARTs in the WaspMote that serve various purposes, one being to connect the micro-controller to the radio modules.

Figure 1: Santander's IoT node and gateway

Radio module

IoT nodes have one XBee 802.15.4 module and one XBee DigiMesh module. Differences between the 802.15.4 and the DigiMesh version are that Digimesh implements a proprietary routing protocol along with more advanced coordination/node discovery functions. In this document, we only consider acoustic data transmission/relaying using the 802.15.4 radio module as the DigiMesh interface is reserved for management and service traffic. XBee 802.15.4 offers the basic 802.15.4 [802154] PHY and MAC layer service set in non-beacon mode. Santander's nodes have the "pro" version set at 10mW transmit power with an advertised transmission range in line-of-sight environment of 750m. Details on the XBee/XBee-PRO 802.15.4 modules can be found in [XBeeDigi] [DMDigi].

Review of the HobNet test-bed hardware

IoT nodes

Sensor nodes in the HobNet test-bed consist in AdvanticSys TelosB motes, mainly CM5000 and CM3000, see figure 4, that are themselves based on the TelosB architecture. These motes are built around a TI MSP430 microcontroller with an embedded Texas Instrument CC2420 802.15.4 compatible radio module. The TelosB description and data-sheet can be found in [TELOSB]. Documentation on the AdvanticSys motes can be found in [ADVAN]. AdvanticSys motes run under the TinyOS system [TINYOS]. The last version of TinyOS is 2.1.2 and our tests use this version.

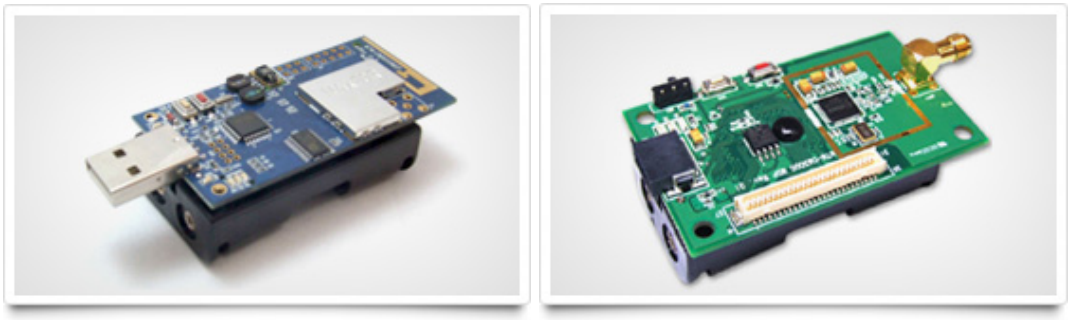


Figure 2: CM5000 (left) and CM3000 (right)

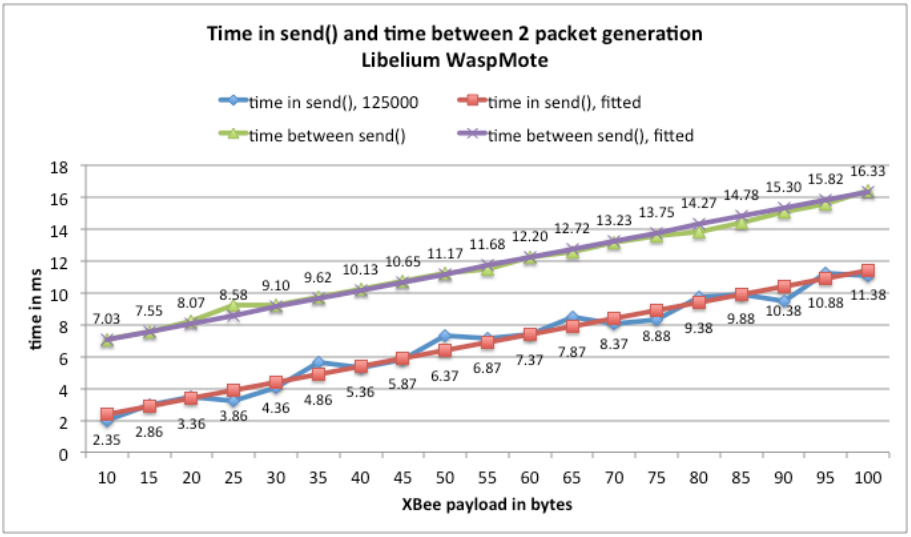
Radio module

The CC2420 is less versatile than the XBee module but on the other hand more control on low-level operations can be achieved. The important difference compared to the previous Libelium WaspMote is that the radio module is connected to the microcontroller through an SPI bus instead of a serial UART line which normally would allow for much faster data transfer rates. The CC2420 radio specification and documentation are described in [CC2420].

The TinyOS configuration by default uses a MAC protocol that is compatible with the 802.15.4 MAC (Low Power Listening features are disabled). It also uses ActiveMessage (AM) paradigm to communicate. As we are using heterogeneous platforms we will rather the TKN154 IEEE 802.15.4 compliant API. We verified the performances of TKN154 against the TinyOS default MAC and found them greater.

Review of maximum IoT sending performance

Regarding the network indicators we already reported in deliverable 1.1 the time spent in a generic send() function, 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 developed a traffic generator with advanced timing functionalities. Packets are sent back-to-back with a minimum of data manipulation needed to maintain some statistics (counters) and to fill-in data into packets, which is the case in a real application. On the WaspMote, we increased the default serial baud rate between the microcontroller and the radio module from 38400 to 125000. The Libelium API has also been optimized (for instance, we also remove the overhead of waiting for transmission status, which is not very relevant for real-time acoustic data) to finally cut down the sending overheads by almost 3 compared to the original Libelium API! Figure 3(top) shows t_{send} and t_{pkt} for the WaspMote. Results for AdvanticSys TelosB are shown in Figure 5(bottom).



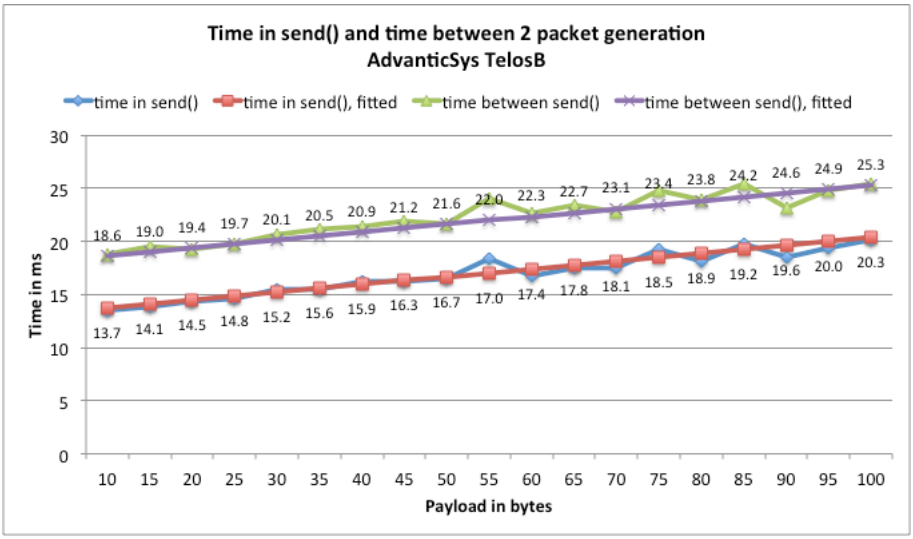


Figure 3: t_{send} and t_{pkt} for for WaspMote (top) and AdvanticSys TelosB (bottom)

Review of maximum IoT relaying performance

We also used the traffic generator to send packets to a receiver where we measured (i) the time needed by the mote to read the received data into user memory or application level, noted t_{read}, and (ii) the total time needed to relay a packet. Figure 4 shows the results.

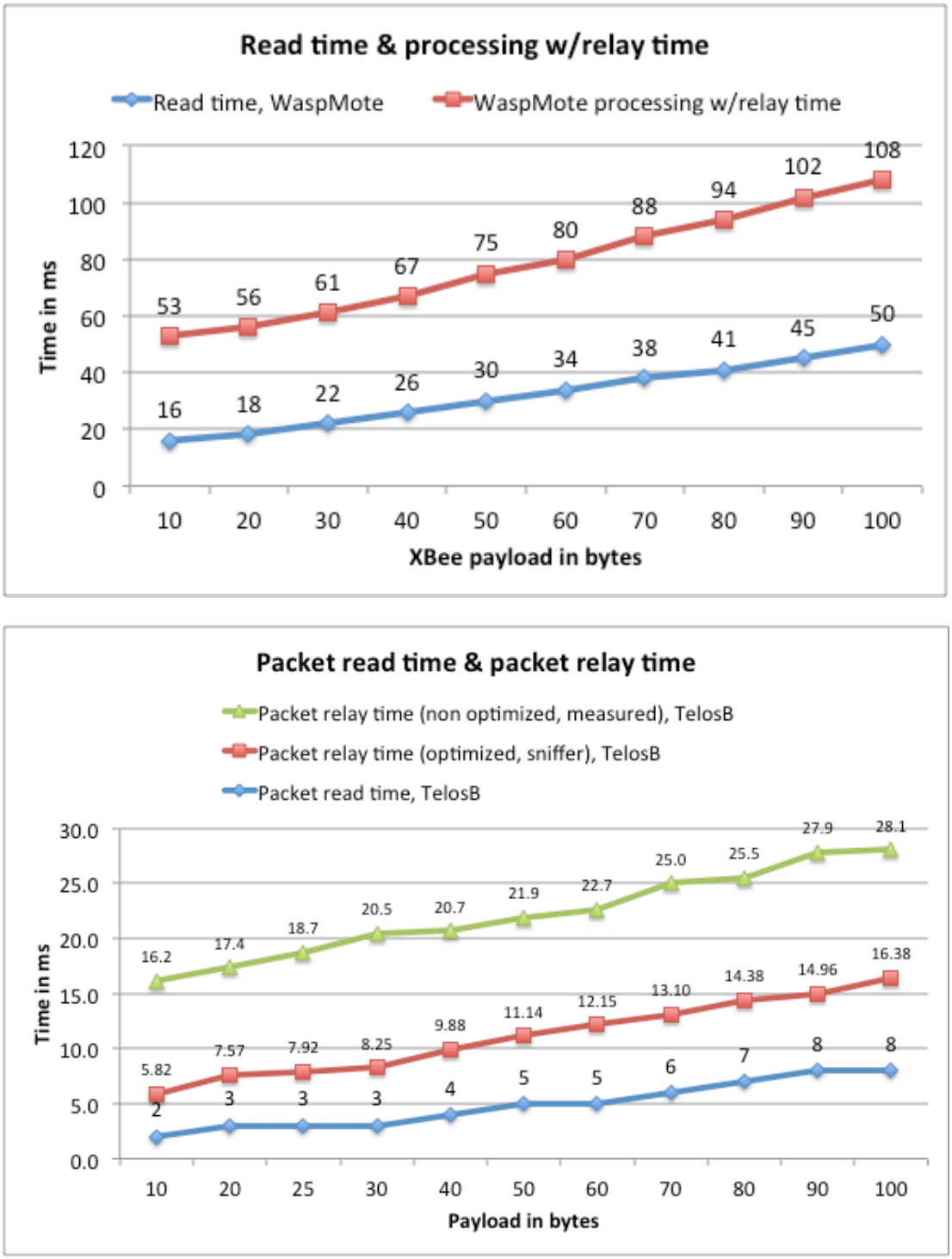


Figure 4: t_{read} and t_{relay} for for WaspMote (top) and AdvanticSys TelosB (bottom)

Review of minimum requirements at sender and relay nodes in a multi-hop environment

Minimum requirements at the sender side

Codec	Minimum sending rate
Raw 4KHz 8KHz	 100 bytes every 25ms 100 bytes every 12.5ms
Speex 8000bps A1 A2 A3 A4	 24 bytes every 20ms 48 bytes every 40ms 72 bytes every 60ms 96 bytes every 80ms
Codec2 2400bps A1 . . An (1≤n≤11) 3200bps A1 . . An (1≤n≤9)	 9 bytes every 20ms . . 9*n bytes every n*20ms 11 bytes every 20ms 11*n bytes every n*20ms

Table I: summary of the minimum requirements at the sender side

Buffer size & packet drop relationship at relay nodes

Time before packet drop due to a full receive buffer

WaspMote audio, WaspMote & TelosB relay nodes				
Q	4KHz/W	8KHz/W	4KHz/T	8KHz/T
1000	0.33	0.14	2.33	0.23
1500	0.49	0.21	3.50	0.34
2000	0.65	0.28	4.67	0.45
2500	0.81	0.35	5.83	0.56
3000	0.98	0.42	7.00	0.68
3500	1.14	0.49	8.17	0.79
4000	1.30	0.57	9.33	0.90
4500	1.46	0.64	10.50	1.02
5000	1.63	0.71	11.67	1.13

TelosB audio board, WaspMote relay node				
Q	A1	A2	A3	A4
1000	1.27	1.81	2.56	3.40
1500	1.91	2.72	3.84	5.10
2000	2.54	3.63	5.11	6.79
2500	3.18	4.53	6.39	8.49
3000	3.82	5.44	7.67	10.19
3500	4.45	6.35	8.95	11.89
4000	5.09	7.25	10.23	13.59
4500	5.72	8.16	11.51	15.29
5000	6.36	9.07	12.79	16.99

Table II: time before packet drop due to a full receive buffer

Maximum supported packet loss rate

Codec	Maximum packet loss rate for speech understanding
Raw 4KHz & 8KHz	50%
Speex 8000bps	35%
Codec2 2400bps 3200bps	20% 30%

Table III : summary of the maximum packet loss rate for speech understanding

3. Audio streaming and developed hardware

Motivations & purposes of audio streaming

The EAR-IT project is one of these projects which focuses on large-scale "real-life" experimentations of intelligent acoustics for supporting high societal value applications and delivering new innovative range of services and applications mainly targeting to smart-buildings and smart-cities.

Since the beginning, we faced challenges as Internet of things devices are known for their limited processing capability and also their limited autonomy. Furthermore it was obvious that wireless network will not allow large transmission and will have limitation in bandwidth (which was confirmed by D1.1 and D1.2). However EAR-IT decided still to cope with some challenges and do some experiment to still explore the possibility to ask regular simple device to provide audio streaming with state of art coding techniques and thanks to our acoustic expert. In exploring some possibility we will also explore possibility to enable audio to any FIRE/test facility and give the condition to use audio (This is the del 1.2).

To conduct our research and experiment we had to consider the technical capability of audio streaming but then also to look at the foreseen potential application. One of the audio scenarios that we considered in EAR-IT is the audio streaming possibility where audio samples can be captured on an on-demand basis by an IoT node and streamed in a near real-time fashion to a control command centre under the supervision of a human operator. Motivations are to better understand an emergency situation with audio information from people on the emergency scene.

Basic principles & constraints

Acoustic data are usually obtained through a sampling process of an analog signal from a microphone. Narrow-band sampling processes use a sampling rate lower than 8KHz while wide-band sampling usually samples at a frequency of at least 16KHz. An A/D converter usually performs the sampling process providing the digital samples on a number of bits, e.g. a digital sample on 10 bits gives values between 0 and 1023 for instance. Sampling at 8KHz means that the A/D converter must provide 1 sample every 125us.

Most of audio processes used in communication networks are narrow-band audio with a sampling rate equal or lower than 8KHz. Also, samples are usually coded on 8 or 16 bits, meaning that the digital value provided by the A/D converter is usually mapped (quantization stage) on 8 or 16 bits. Therefore, in the so-called raw format, the continuous flow of audio data represents an 64kbit/s data flow if samples are 8-bit wide: 8*8000=64000 bits. The various steps towards digitized audio are depicted in the next figure below: from sampling to quantization to obtain digitized audio.

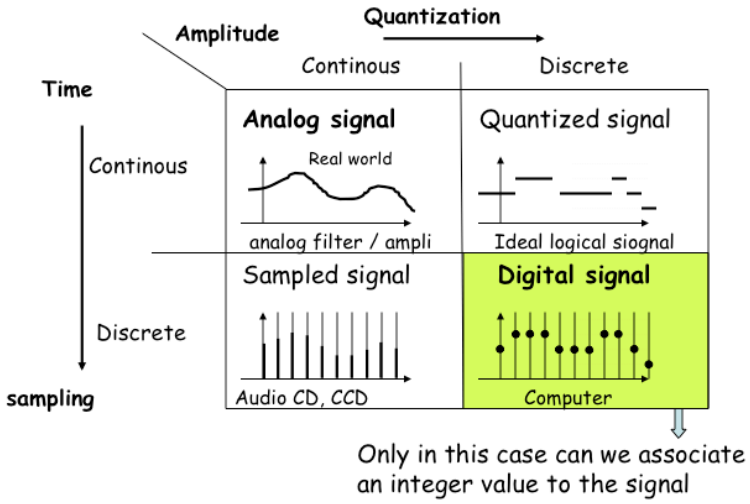


Figure 5: digitized audio

In the EAR-IT project, the hardware limitations of IoT nodes impose the use of narrow-band audio with sampling rates smaller or equal to 8KHz. Also, the limitations on the sending rate at the application level and on the radio bandwidth generally discard audio bit rates greater than 64kbps as pointed out in the EAR-IT deliverable 1.1 on the network qualification.
The raw audio can be compressed in various manners and many compression algorithms have been proposed and used widely in communication networks and applications: traditional wired telephony systems, Voice over IP, GSM, … Compression can provide a much smaller bit rate to adapt the required throughput to the available bandwidth of the transmission system. This is particularly important for near real-time audio in streaming applications. The term "audio codec" will then be used as a generic term to designate one audio compression scheme. There are hundreds of different audio codecs used in the telephony, music and video industry to name them all. Although not an authoritarian source, a quite exhaustive list of audio codecs and audio containers are presented on http://en.wikipedia.org/wiki/List_of_codecs and http://en.wikipedia.org/wiki/Comparison_of_container_formats.

Once audio has been digitized into 8-bit samples, compressed and grouped into a number of samples for transmission, near real-time audio streaming usually needs small packet jitter in order to avoid gaps in the audio playout. As bounded jitter is difficult to achieve because timing guarantees are difficult to ensure in communication protocols at low cost, a best-effort approach is commonly used with an intermediate playout buffer. Figure 6 below illustrates the basic principles of a playout buffer with the objective of shaping and regulating the packet output rate.

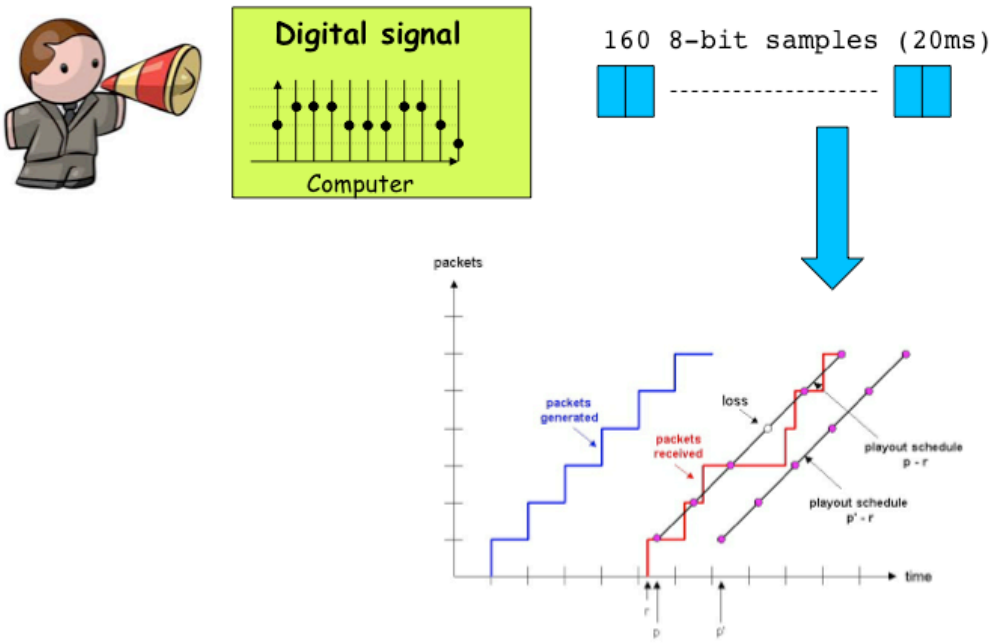


Figure 6: playout buffer to handle packet jitter

Raw audio with 8kHz sampling on Libelium WaspMote

Most of IoT nodes are based on low speed microcontroller (Atmel 1281 at 8MHz for the Libelium WaspMote and TI MSP430 at 16Mhz for the AdvanticSys) making simultaneous raw audio sampling and transmission nearly impossible when using only the mote microcontroller.

To leverage these performance issues, one common approach is to dedicate one of the 2 tasks to another microcontroller:

1. Use another microcontroller to perform all the transmission operations (memory copies and buffering, frame formatting, among others);

2. Use another microcontroller to perform the sampling operations (generates interruptions, reads analog input, performs A/D conversion and possibly encodes the raw audio data).

Our first hardware development is based on the first solution. A Libelium WaspMote is equipped with an amplified microphone and the host microcontroller has the task of periodically sampling the noise level. The XBee radio module which has an embedded internal microcontroller is configured to handle all the sending operations when running in so-called transparent mode (API mode 0 of XBee module). Figure 7 shows the Libelium WaspMote hardware.

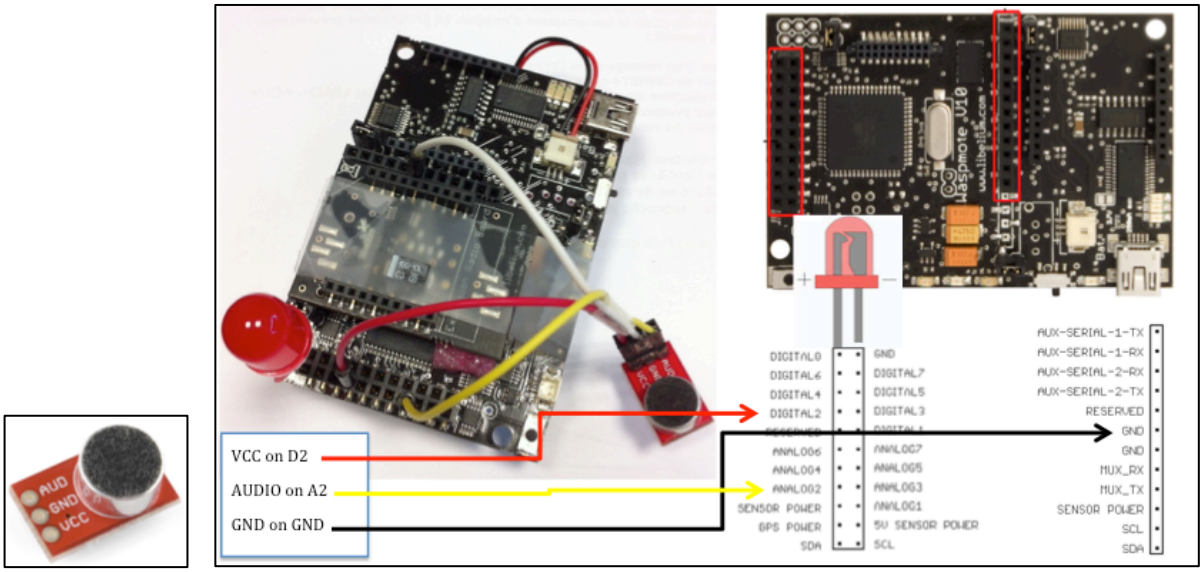


Figure 7: raw audio capture with Libelium WaspMote

Description

1. Use a pre-amplified MIC and connect it a analog input of the Libelium WaspMote. We use the following MIC: http://www.cooking-hacks.com/shop/sensors/sound/breakout-board-for-electret-microphone (see figure3, left) and connect it to the WaspMote (AUD to Analog2, VCC to Digital 2 to get 3.3V and GND to GND, see figure 3, right).

2. Configure an XBee radio module in transparent mode (API mode 0). Broadcast or unicast communications can be used but this has to be configured prior to sending any data because we let the XBee microcontroller do all the sending tasks. Here is a text taken from the XBee manual from Digi:

   « When operating in this mode, the modules act as a serial line replacement - all UART data received through the DI pin is queued up for RF transmission »

   « Data is buffered in the DI buffer until one of the following causes the data to be packetized and transmitted:

   a. *No serial characters are received for the amount of time determined by the RO (Packetization Timeout) parameter. If RO = 0, packetization begins when a character is received.*
   b. *The maximum number of characters that will fit in an RF packet (100) is received.*
   c. *The Command Mode Sequence (GT + CC + GT) is received. Any character buffered in the DI buffer before the sequence is transmitted. »*

   In our case, data will be sent by the XBee radio module internal microcontroller either on case (a) or (b).

3. Sample the analog input (Analog2) at 4KHz or 8KHz, i.e. read analog value once every 250us or 125us. A/D converter gives a 10-bit sample so it has to be converted into an 8-bit sample.

4. As the XBee radio module is connected to the host microcontroller, i.e. the Atmel 1281, with a serial UART line, we can just write in a dedicated register the 8-bit sampled value.

5. Receive on a PC or a gateway (Libelium Meshlium for instance) using an XBee radio module in AP0 mode that will send data to the PC serial interface.

6. Continuously read PC or gateway serial port and send data to standard output (usually `stdout` on a Unix machine). Use redirection to inject `stdout` into an audio player such as `play` (part of `sox` package on a Linux machine).

The resulting audio bit stream throughput is 64kbps. At the audio source side, the hardware is capable to sending at that rate because the XBee embedded microcontroller handles all the framing tasks.

Limitations

Audio streaming is challenging on a multi-hop manner on low-resource IEEE 802.15.4 IoT nodes because of relaying overheads. Figure 8 below depicts an audio streaming scenario where a continuous flow of audio packets need to be sent wirelessly from the source IoT node to the nearest gateway connected to the Internet.

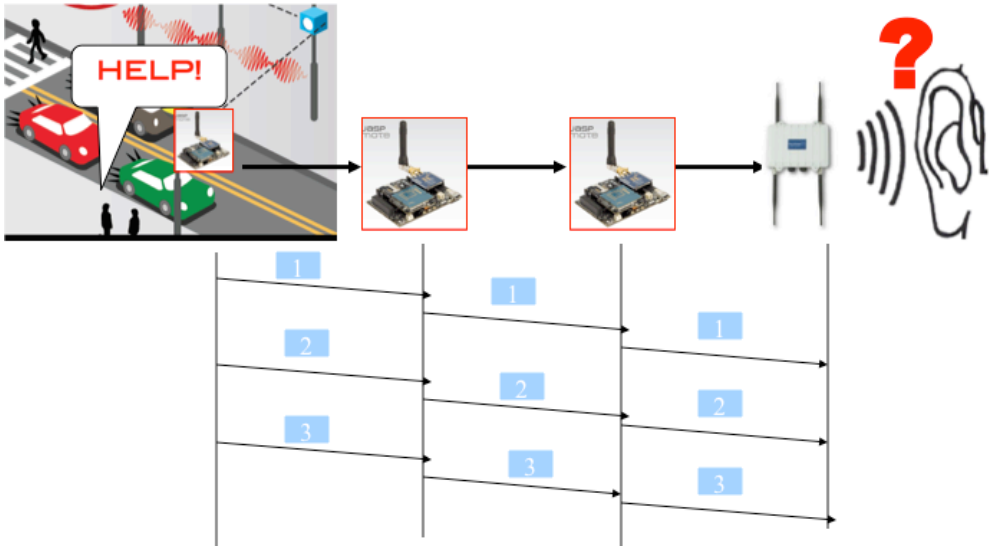


Figure 8: multi-hop audio challenges

Even though a WaspMote can use its XBee module in transparent mode to increase its sending capability, it is not efficient for receiving and relaying incoming packets. In addition, with the XBee radio module configured in transparent mode, it is very difficult to perform multi-hop transmission because the destination address of the next hop needs to be configured on the XBee module prior to packet transmission.

Therefore the solution described above with the Libelium WaspMote can practically be realized only to have 1-hop transmission from the audio source to a gateway, which dramatically reduces the acoustic sensing possibilities.

Development of a dedicated audio board

To overcome all the limitations associated to raw audio, we developed a dedicated audio board to handle the sampling and compression steps. By reducing the audio bit stream throughput, multi-hop audio can be realized by keeping the relaying throughput in the performance range of intermediated nodes (see figure 4).

Regarding the audio compression process, in the EAR-IT project it is important to use open-source codecs to insure the largest dissemination, compatibility and interoperability. Another important criteria is the availability of libraries and tools that can be easily installed, used and integrated on any Linux-box on the market. The minimum requirements therefore greatly depend on the audio codec that will be used.

The audio board, initially developed for the AdvanticSys TelosB can be connected to other mote platforms provided that a serial port (UART) can be used to feed in the encoded audio data. ANNEX.C will describe how the audio board has been successfully connected to a Libelium WaspMote and to an Arduino MEGA 2560. Connecting the audio board to an embedded Linux board such as Rasperry PI or BeagleBone can be done in a straightforward manner with a serial-to-USB adapter and using standard Linux tools/scripts/commands to read the serial port.

Description

1. Develop a daughter audio board with its own microcontroller that will be connected to the AdvanticSys expansion connector. The audio board will handle the sampling operations and encode in real-time the raw audio data into Speex codec (www.speex.org). 8KHz sampling and 8-bit sample will be used to produce an optimized 8kbps encoded Speex stream (speex encoding library is provided by Microchip).

2. The audio board is designed and developed through collaboration with INRIA CAIRN research team. Figure 9 shows a schematic of the audio board design.

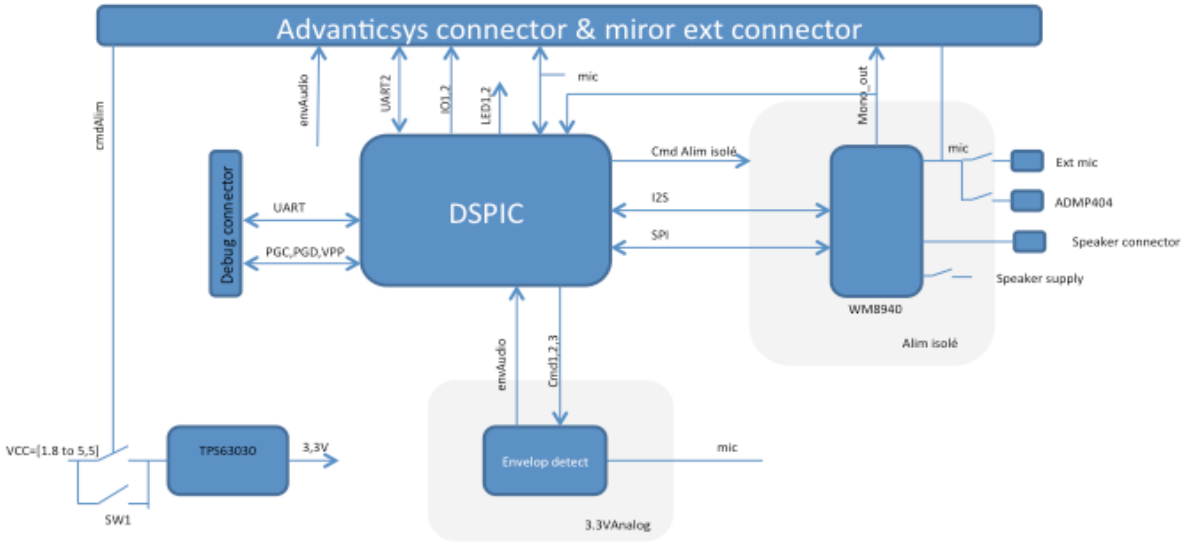


Figure 9: developed audio board schematic

The audio board has a built-in omnidirectional MEMs microphone (ADMP404 from Analog Devices) but an external microphone can also be connected. The microphone signal output is amplified, digitized and filtered with the WM8940 audio codec. The audio board is built around a 16-bit Microchip dsPIC33EP512 microcontroller clocked at 47.5 MHz that offers enough processing power to encode the audio data in real-time. From the system perspective, the audio board sends the audio encoded data stream to the host microcontroller through an UART component. The host mote will periodically read the encoded data to periodically get fixed size encoded data packets that will be transmitted wirelessly through the communication stack.

3. Connect the audio board to the AdvanticSys through the 51-pin expansion connector:

from the system perspective, the audio board sends the audio encoded data stream through an UART connection to the host micro-controller.

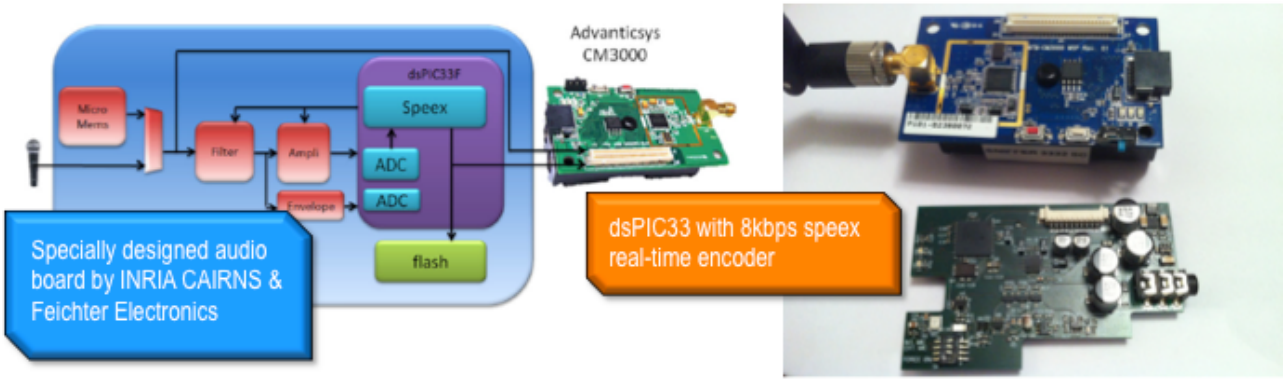


Figure 10: developed audio board and AdvanticSys TelosB with the audio board

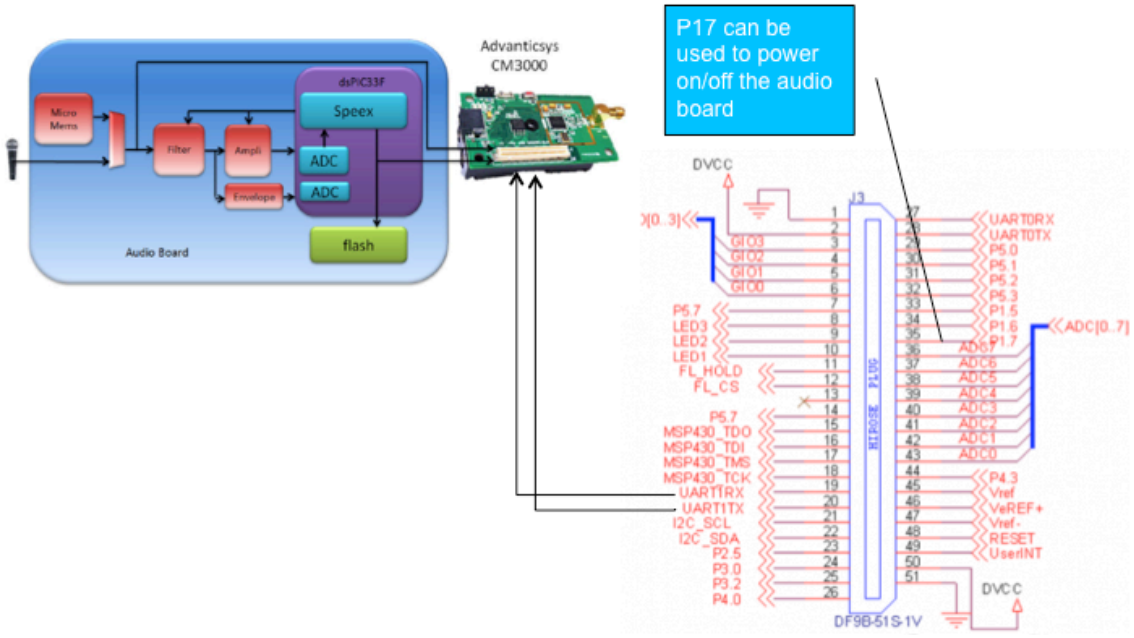


Figure 11: developed audio board connectivity schema on an AdvanticSys TelosB

4. 8KHz `speex` works with 20ms audio frames: every 20ms, 160 8-bit samples of raw audio data are sent to the `speex` encoder to produce a 20-byte audio packet. 2 framing bytes are added and 2 additional bytes are used to store a sequence number and the frame size. The total audio packet is then 24 bytes as depicted by figure 12.

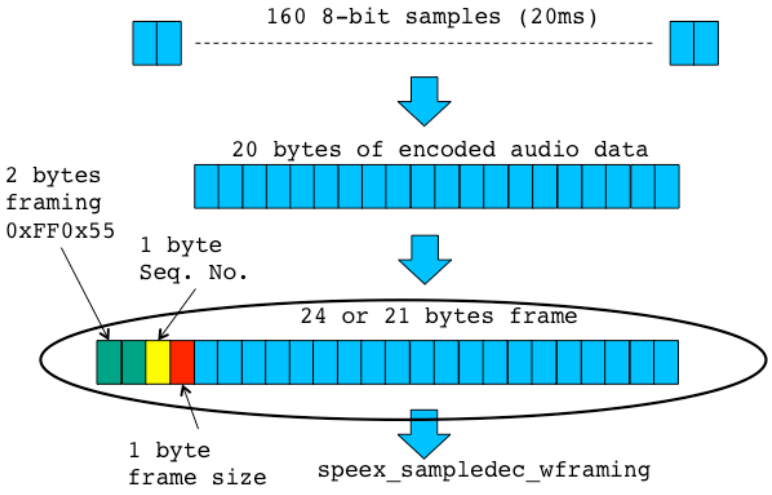


Figure 12: audio frame format, encoded audio data are in speex format

5. Read encoded date from the host mote to periodically get fixed size encoded data packets that will be transmitted wirelessly through the communication stack (provided by TinyOS environment).

6. Receive on a PC or a gateway (Libelium Meshlium for instance) using another AdvanticSys mote as a base station mote.

7. Continuously read PC or gateway serial port and send data to standard output (usually `stdout` on a Unix machine). Use redirection to inject `stdout` into a Speex decoder that will also send on stdout the raw decoded audio data.

8. Use redirection to inject `stdout` into an audio player such as `play` (part of `sox` package on a Linux machine).

Control software on audio sensor mote

The audio board is independent from the host microcontroller. The host mote will periodically read the encoded data (made available on a serial port) to periodically get fixed size encoded data packets that will be transmitted wirelessly through the communication stack.

We implemented some additional features to demonstrate the on-demand multi-hop audio streaming scenario. The control software can receive a number of ASCII commands prefixed by "/@" and ended by "#":

1. **"C"** command to start or stop the audio capture and transmission process

   "/@C1#" starts the capture and "/@C0#" stops the capture

2. **"D"** command to set the destination address (next hop in case of multi-hop). 16-bit or 64-bit IEEE 802.15.4 address can be specified.

   "/@D0100#" or "/@D0013A2004086D82E#"

3. **"A"** command to aggregate a number of audio frames into a radio packet. Possible values are 1, 2, 3, 4 or 6. The 6 value has a special meaning as it will be explained later on in the Multi-hop section. The purpose of audio frame aggregation is to increase the time window for relaying nodes to relay the audio packet.

Speex 8000bps A1 A2 A3 A4 A6	 24 bytes every 20ms 48 bytes every 40ms 72 bytes every 60ms 96 bytes every 80ms 96 bytes every 120ms

Multi-hop audio

Figure 13 illustrates the multi-hop audio streaming scenario. The audio source (0x0090, 16-bit address) is configured to send audio data to relay node 0x0020 which is also configured to relay audio data to sink node 0x0100. Aggregation level can be set at any time, even during an active capture using the A command.

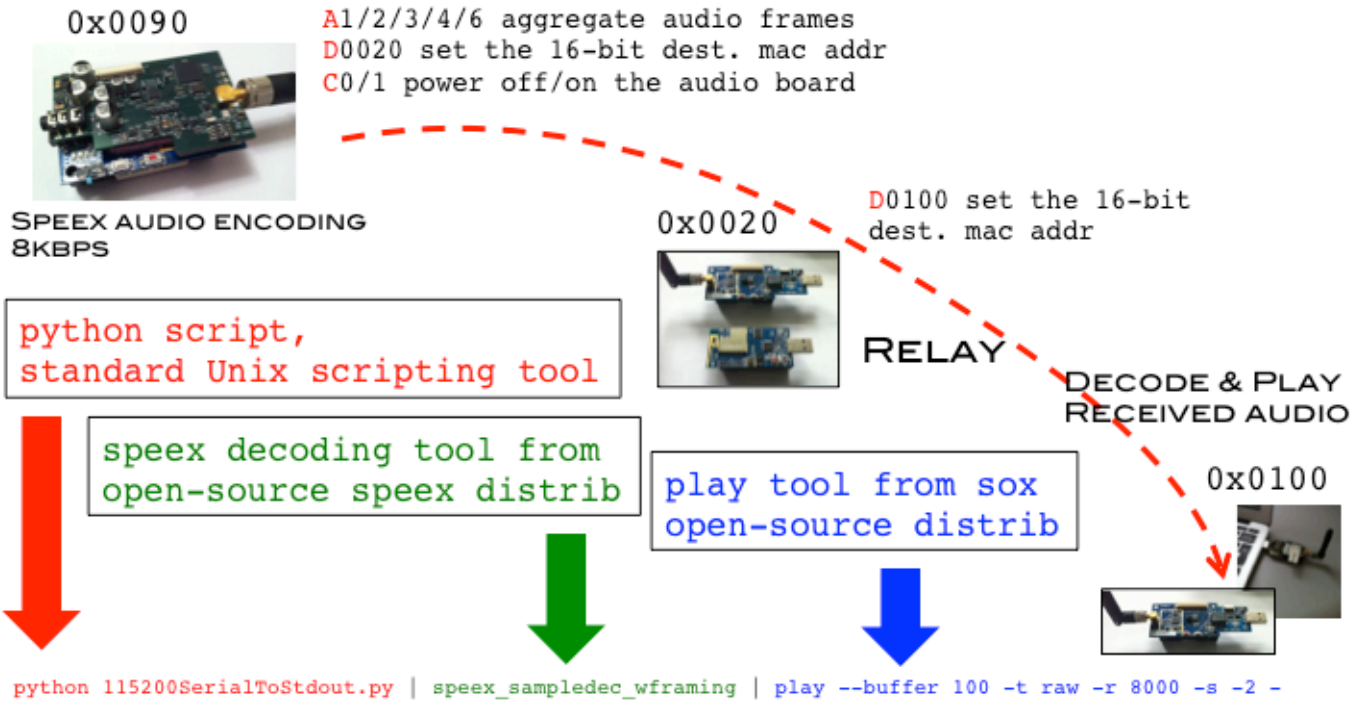
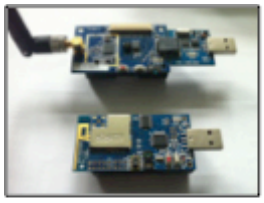


Figure 13: Multi-hop audio streaming scenario with the developped audio board

When using AdvanticSys TelosB relay nodes, the relaying performance of an optimized version is sufficient to handle a 24-byte packet every 20ms as shown in figure 14 below. Previous version of relay nodes required A2 aggregation as relaying a 24-byte packet needed in average about 19ms. However, this relaying time can be greater than 20ms in many cases, causing packet drops at the relay nodes.



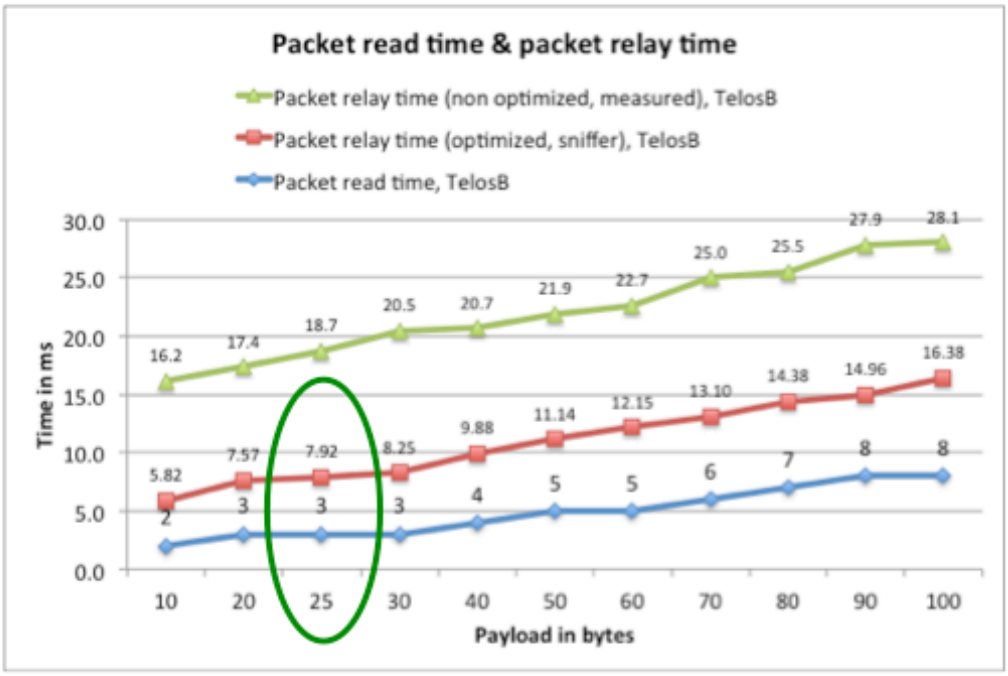


Figure 14: TelosB relaying performances, no need for aggregation

On the Santander's SmartSantander test-bed, the relay nodes are Lebelium WaspMote which has higher relaying overheads. In this case, even A4 aggregation mode can not provide a sufficient time window for the relay node as depicted by figure 15 below.

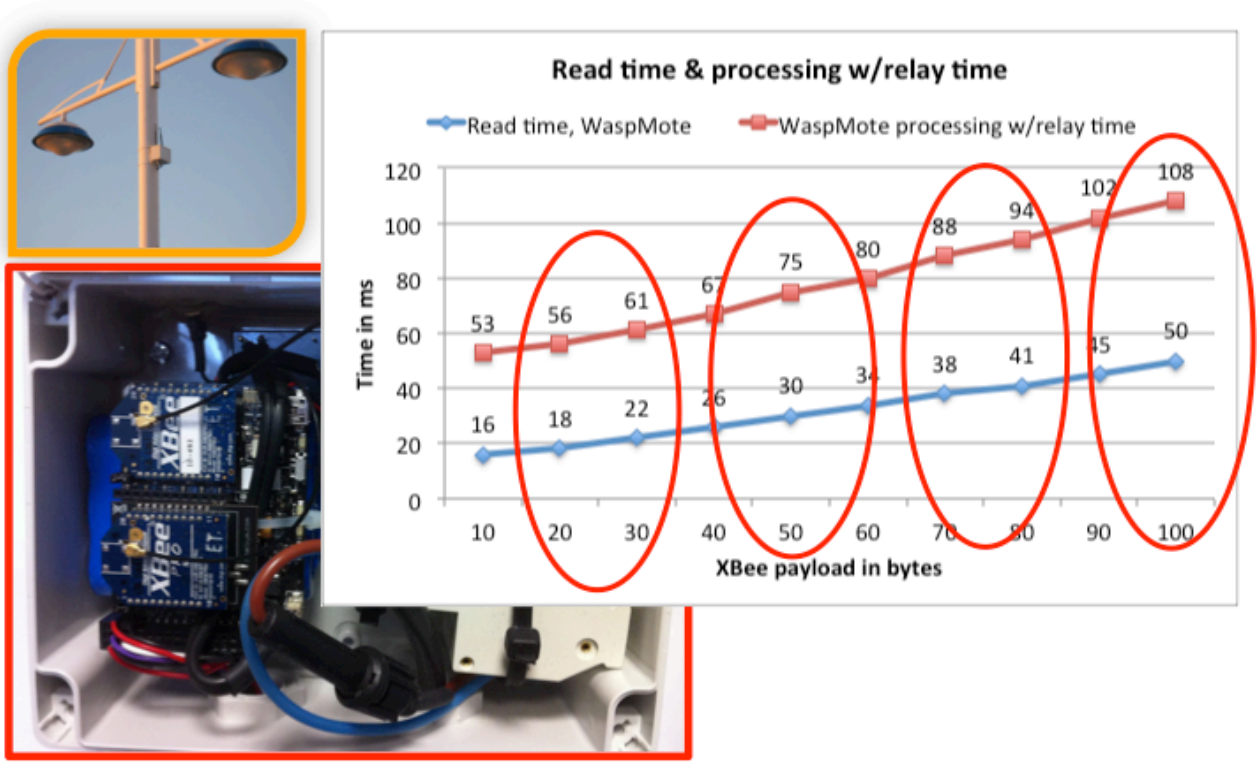


Figure 15: WaspMote relaying performances, need specific aggregation mode

In order to provide multi-hop audio streaming on slow IoT nodes, it is necessary to discard a number of audio frames at the source. This is the purpose of the special A6 aggregation mode: 6 audio frames are captures to provide a 120ms time window but only 4 audio frames are transmitted. This behavior is illustrated in figure 16.

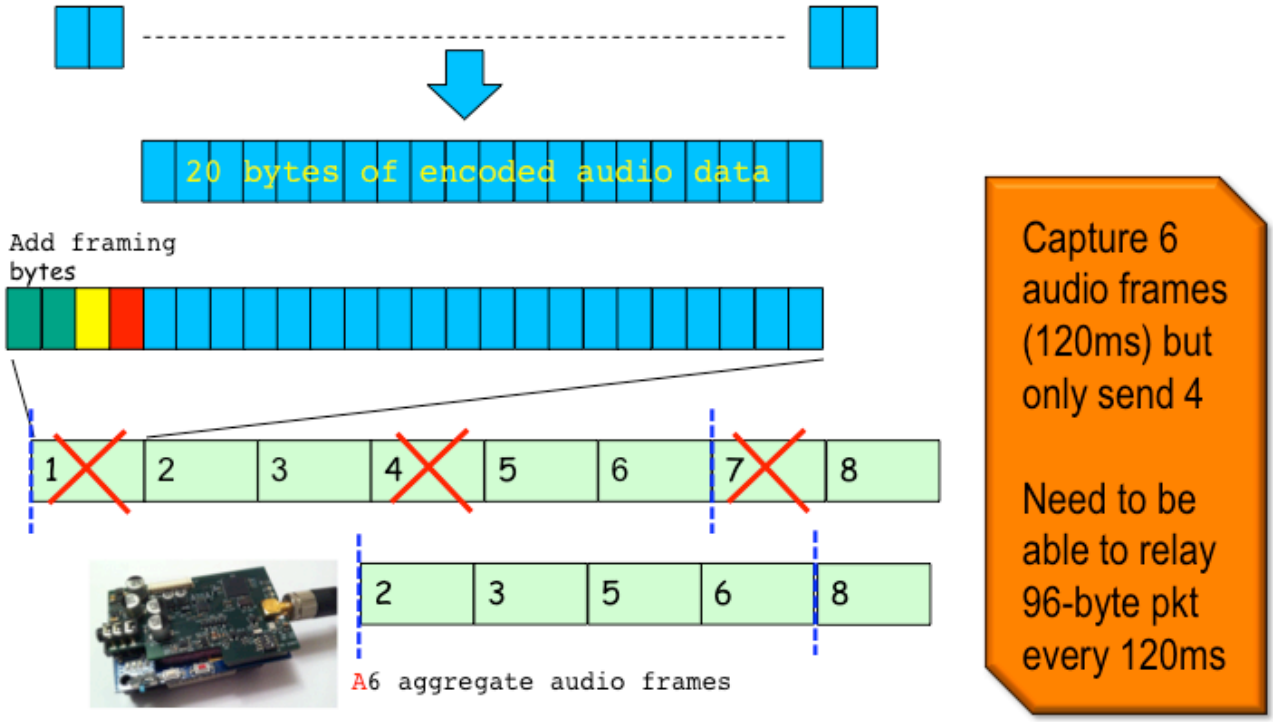


Figure 16: A6 aggregation mode for slow relay nodes

4. Benchmark methodology and tools

Methodology

In previous deliverable 1.2 we defined some selected performance indicators and presented the minimum requirements for use of acoustic sensors on the various EAR-IT test-beds based on WSN and IoT nodes with IEEE 802.15.4 radio technology. These performance indicators were categorized into:

1. Network performance indicators (NETWORK)
2. Audio quality indicators (AUDIO),
3. Energy indicators (ENERGY).

The audio quality indicators have already been presented and discussed in previous deliverable 1.2. In this document we will measure experimentally the network performance indicators and the energy indicators on the two EAR-IT test-beds, i.e. Santander's SmartSantander and Geneva's HobNet. Figure 17 illustrates from the source to the destination the various multi-hop constraints and limitations that will impact the audio transmission.

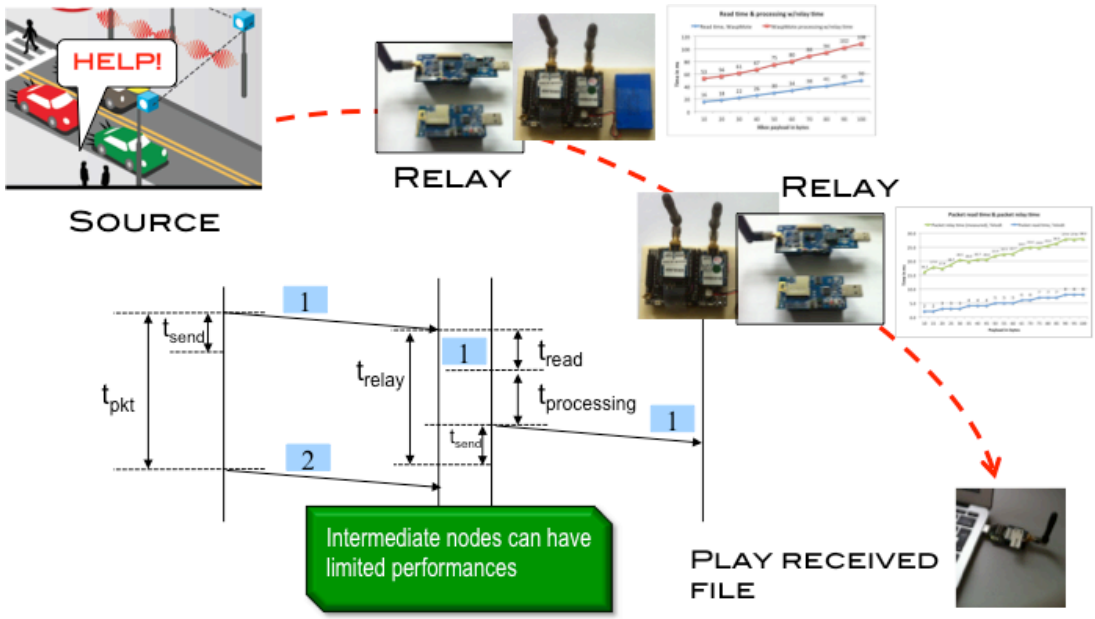


Figure 17: multi-hop constraints and limitations

For network indicators, we will measure:

1. Packet jitter at the source
2. Packet loss rates at 1-hop
3. Packet loss rates at 2-hop
4. Packet relaying time at relay nodes
5. Packet relaying jitter at relay nodes

For energy indicators, we will measure:

1. Energy consumption at the audio source
2. Energy consumption at the relay nodes

Packet analysis tools

The main tool that we will use is the `wireshark` packet analysis tool. We developed a promiscuous packet sniffer with an AdvanticSys TelosB mote that can be connected to `wireshark` in order to display captured frames and get timestamped data on packets that are captured. `wireshark` will allow us to use frame reception time to visualize packets for statistic collection such as transmission latencies and frame jitter. Using the IEEE 802.15.4 frame sequence number we can also obtain packet loss patterns and derive the packet loss rate. Figure 18 below shows an illustration of the packet sniffer and the wireshark tool.

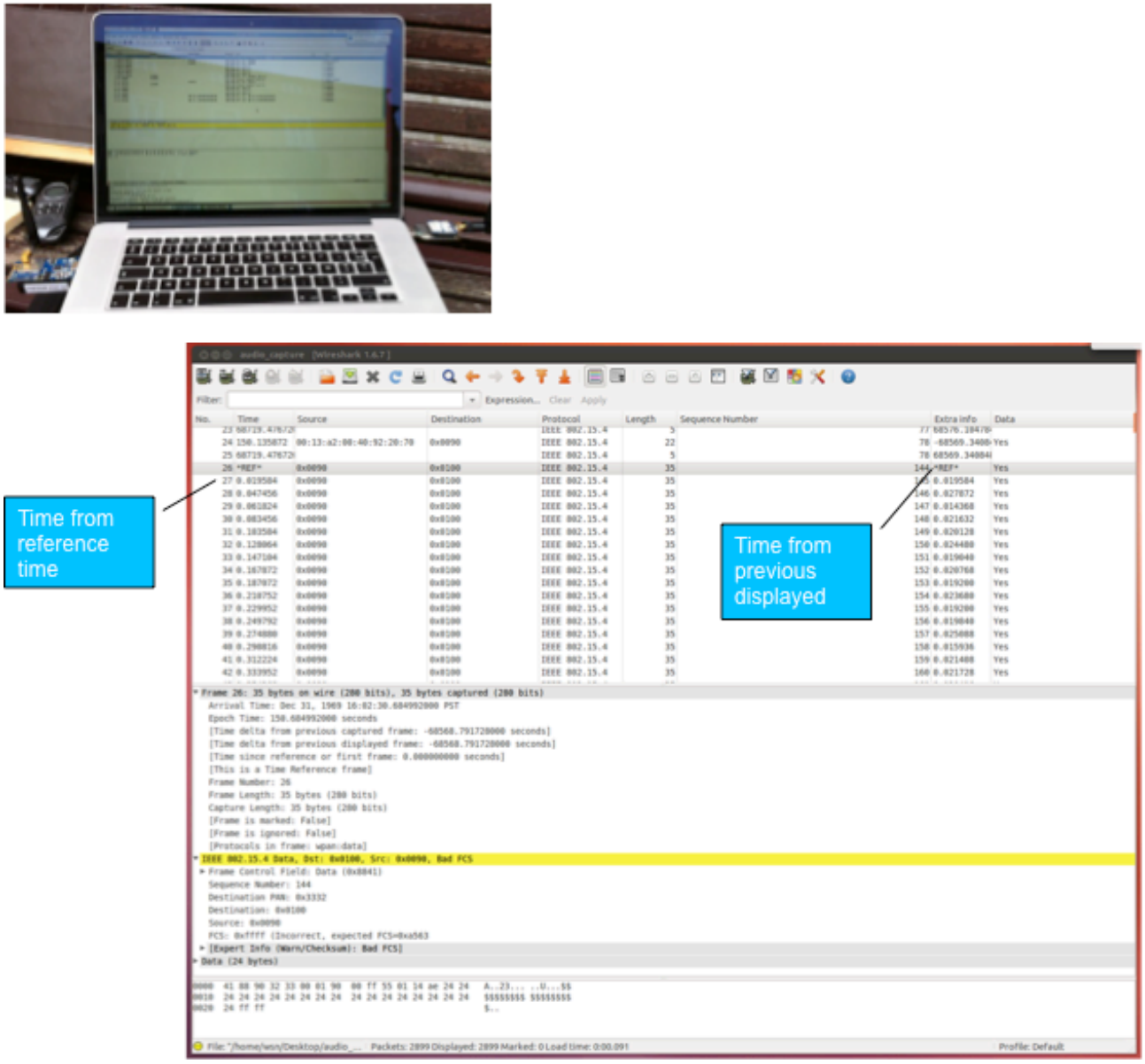


Figure 18: packet analysis tool

5. Network performance indicators

In all the tests described here, the transmission power is set to the maximum radio module power (on the CC2420 of the AdvanticSys TelosB, TinyOS sets the transmission power by default to 0dBm) or to the maximum allowed transmission power (in the case of XBee Pro module for instance on the Libelium WaspMote the European regulation sets the maximum transmission power to 10dBm). In addition, we chose to disable MAC level retransmission in order to highlight packet losses.

4.1 Tests in Santander

All the tests described in this section have been performed in the Santander city during the test campaigns on Feb, 11th and Feb 12th, 2014. 3 locations have been selected. They are identified in figure 19. Gateways (Meshlium) are identified with a red rectangle.

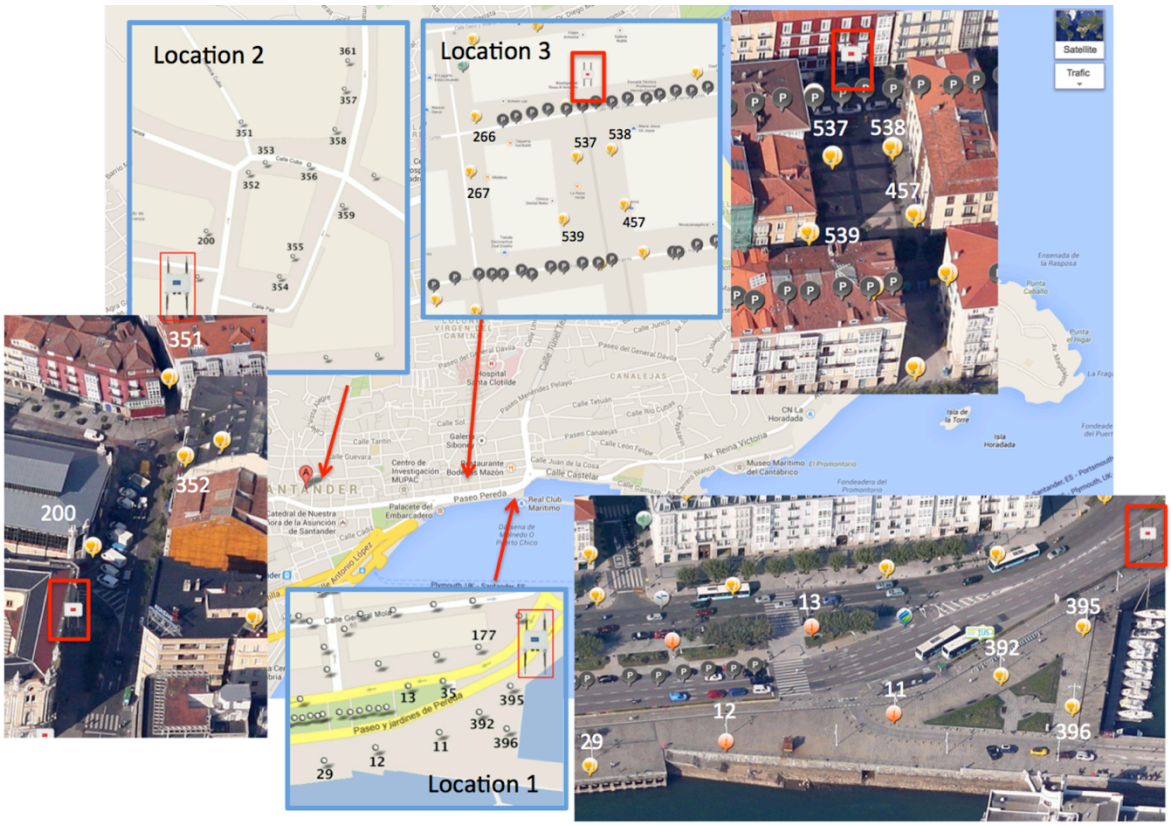


Figure 19 : test locations in Santander (Santander map from Google Maps)

Location 1 is an open-space location on the marina. It has been selected for line-of-sight transmissions. Location 2 is a very dense, central location. It has been selected for tests of non line-of-sight transmission because there are many buildings. Location 3 is a small urban place surrounded by apartment buildings. It has been selected to tests the impact of interference traffic in a typical urban location. In total, we performed 11 tests:

- Location 1: test #1 … test #5
- Location 2: test #6 … test #10
- Location 3: test #11

The tests are also divided into 1-hop and 2-hop transmission:

- 1-hop: test #1, #2, #3, #6, #7, #8, #11
- 2-hop: test #4, #5, #9, #10

1-hop, source to destination

Packet inter-arrival time and packet jitter, line-of-sight transmission

Test #1 is at Santander's location 1 and we measured the packet inter-arrival time from an 8KHz raw audio WaspMote (see figure 7) is placed at location 392 to its associated gateway (Meshlium). The test is depicted in figure 20. This testswill also allow us to measure the packet loss rate in order to predict the audio quality based on the study presented in the previous section. There are no 802.15.4 interference traffic on the radio channel that we selected (channel 18).

The audio WaspMote is programmed with a 30s cyclic ON-OFF behavior. Each period is 15s long. During ON period, the mote captures and sends raw audio data: a 100-byte packet every 12.5ms (100*125us=12.5ms). Thus, 15s of audio generates 1200 packets, one every 12.5ms.

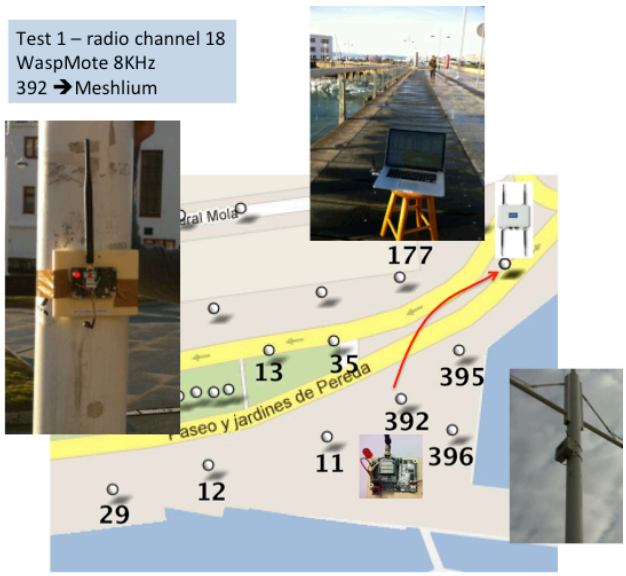


Figure 20 : test #1 at location 1

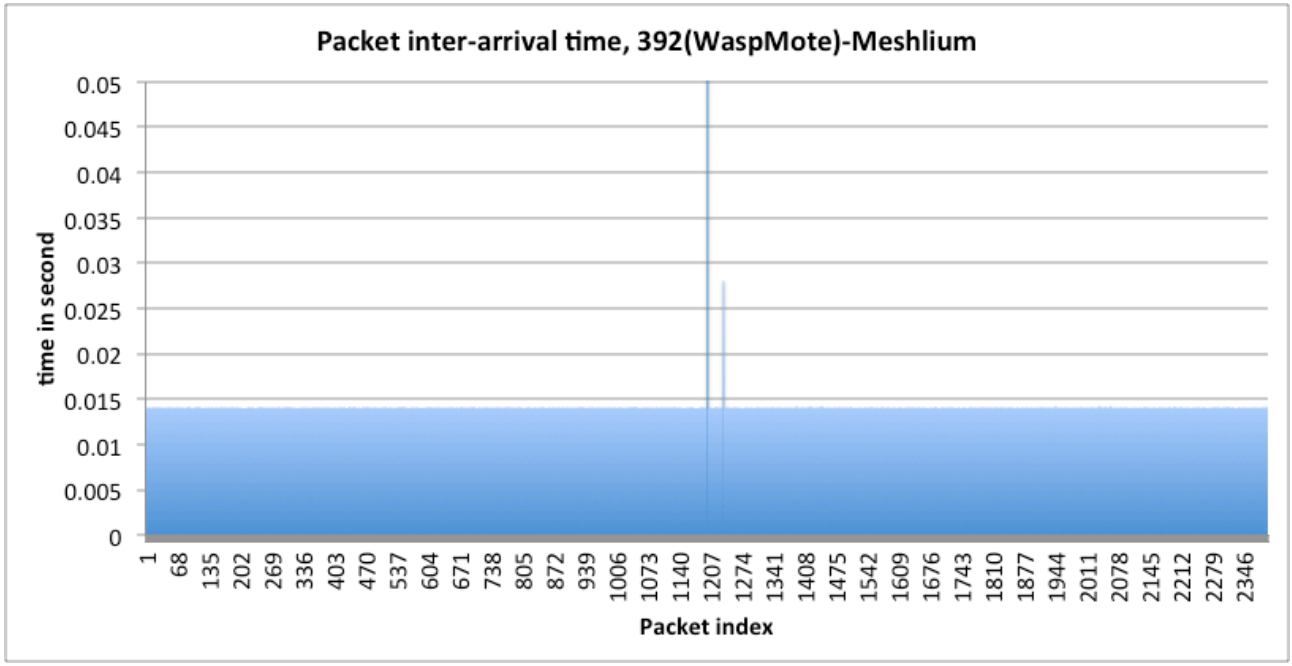


Figure 21 : packet inter-arrival time, test #1

Figure 21 shows the packet inter-arrival time of 2 ON periods of the audio WaspMote. The line at the middle of the graph is the OFF period. The mean inter-arrival time is 0.0139s with a standard deviation of 0.0001176. We can see that packet jitter at 1-hop is very low. In addition, we observed only 1 lost packet out of a total of 2400.

Test #2 consists in a longer transmission distance where the receiver is placed at location 29, see figure 22.

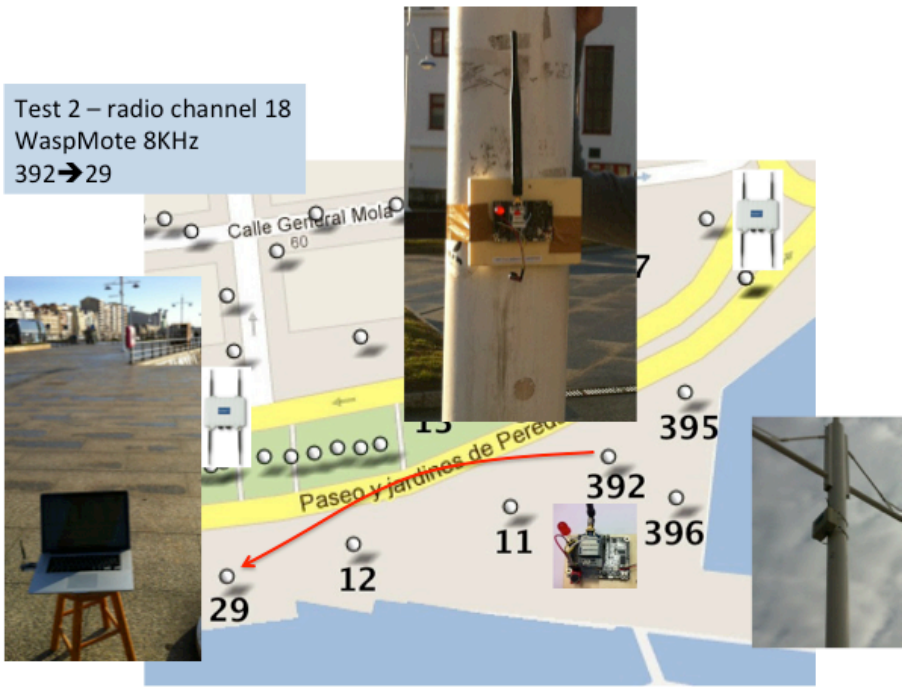


Figure 22 : test #2 at location 1

Figure 23 shows the packet inter-arrival time of 2 ON periods of the audio WaspMote. Here, we observed only 4 lost packets out of a total of 2400.

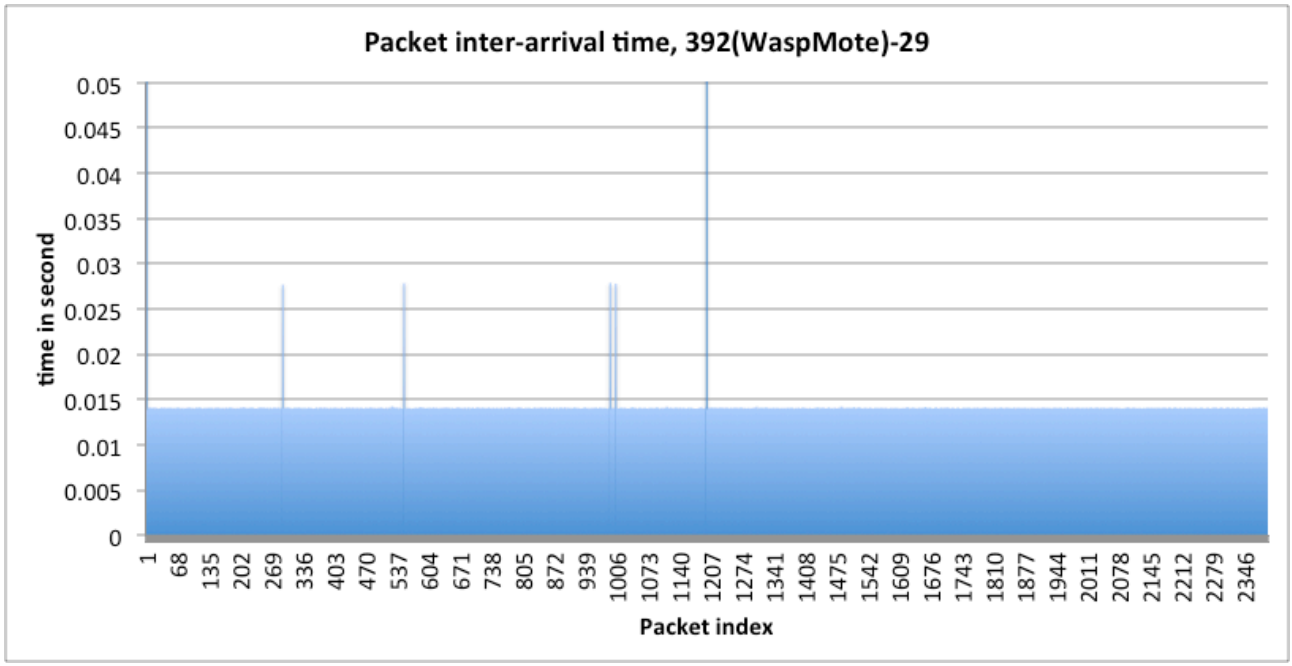


Figure 23 : packet inter-arrival time, test #2

Test #3 now uses the developed audio board plugged into an AdvanticSys TelosB Mote. Figure 24 shows the TelosB with audio board placed on location 392 and sending audio data to the Meshlium at 1-hop.

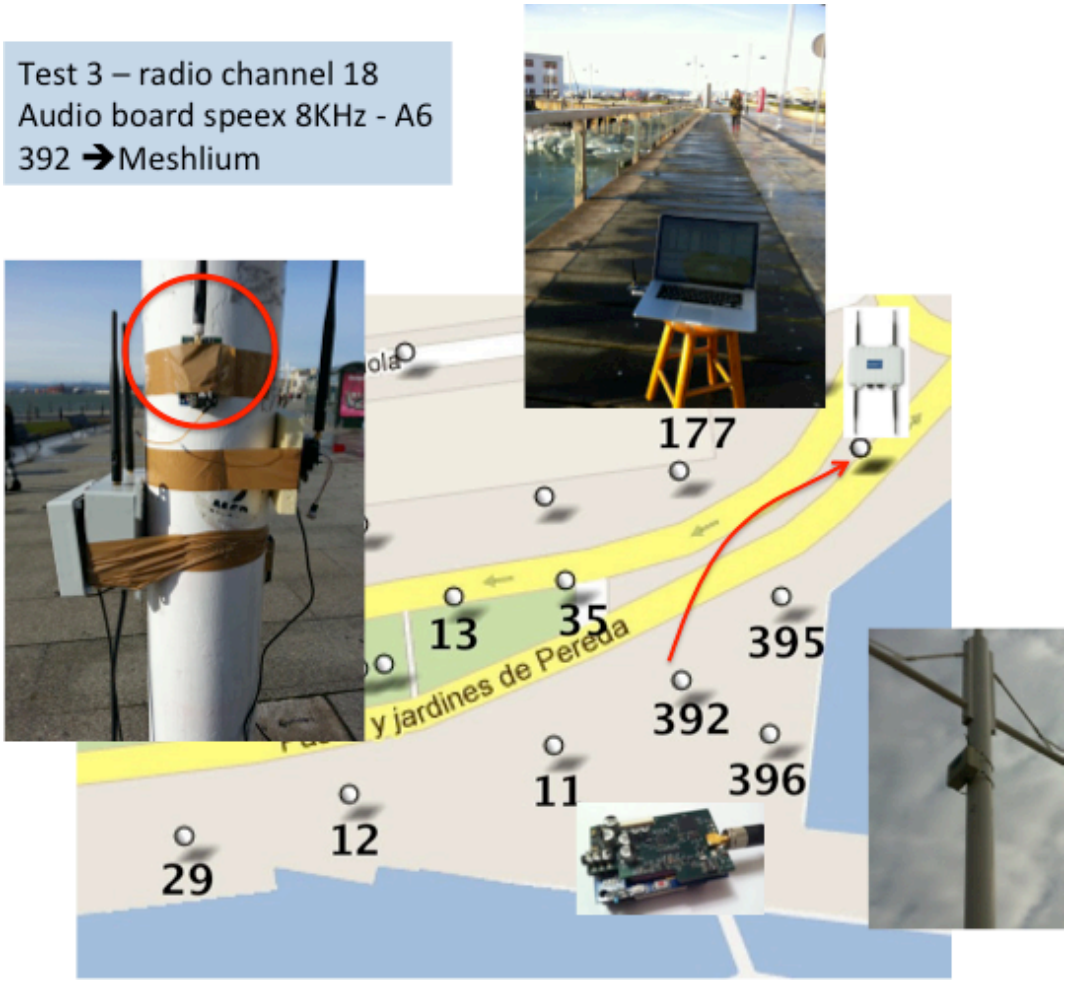


Figure 24 : test #3 at location 1

We use a specific aggregation mode, so-called A6, in this test because relay nodes based on Libelium WaspMote have limited relaying performances as shown previously in figure 4(top). Even if test #3 is a 1-hop test, we wanted to have the same configuration than the multi-hop test that will be described later in this document.

A6 aggregation mode captures 6 audio frames but only send 4 of them in a single 96-byte radio packet. This behavior was depicted in figure 16. Capturing 6 audio frames provides a time window of 6*20ms=120ms. This is required for Libelium WaspMote relay nodes as it will be explained in more details in the multi-hop test section. The important information here is that the audio source sends a packet every 120ms and that the total payload of the packet is 96 bytes.

The AdvanticSys TelosB with the audio board is programmed to start/stop capture and transmission on an on-demand basis. Figure 25 shows the packet inter-arrival time during a 30s (approximately) audio capture . Here, we observed 27 lost packets out of a total of 234 (giving a packet loss rate of about 11.53%). Note that the transmission power of the CC2420 radio module of the TelosB is lower than the one of XBee module (0dBm against 10dBm).

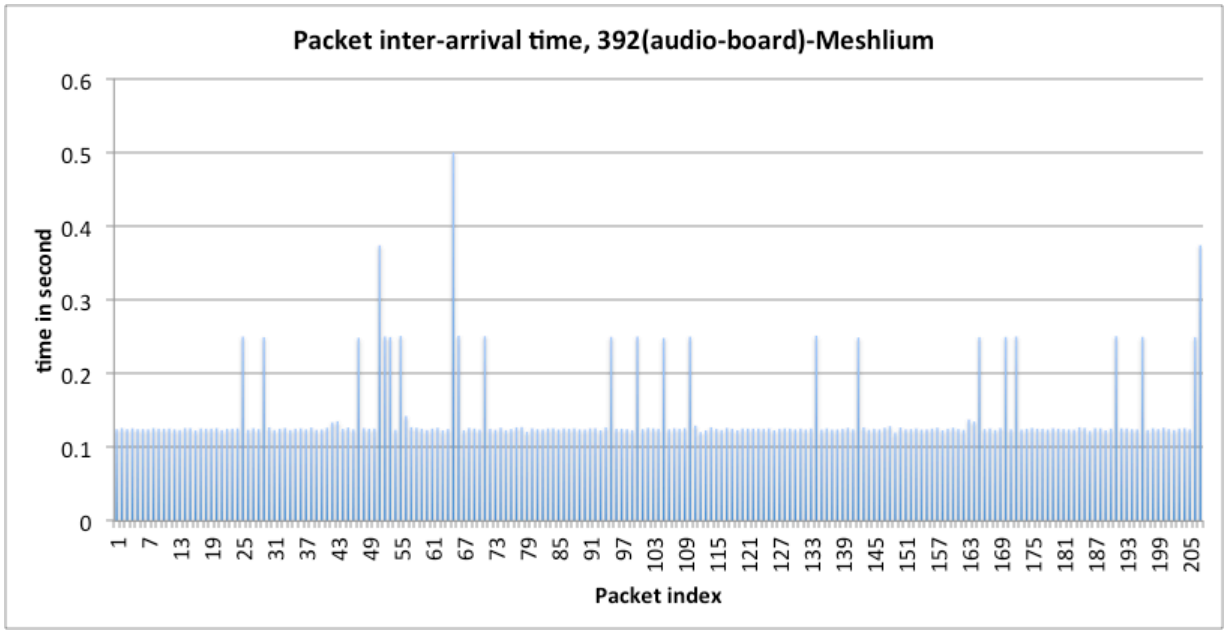


Figure 25 : packet inter-arrival time, A6 level, test #3

If we take the mean from packet 1 to packet 24 (before the first packet loss) then we have a mean inter-arrival time of about 0.1247s with a standard deviation of 0.000879. The packet jitter is then, once again, very small.

In **test #6**, we used Santander's location 2 and we placed the audio WaspMote 8KHz at location 352. The receiver is placed at the Meshlium depicted in figure 26. Although there is no occulting buildings, it is not an open space as there are many people and parked cars in the street.

Figure 26 : test #6 at location 2

Figure 27 shows the packet inter-arrival time of 2 ON periods of the audio WaspMote. The line at the middle of the graph is the OFF period. We observed 423 lost packets in the first ON period (35.35%), with several packet loss bursts, and 204 lost packets in the second ON period (17%). Each period have a total of 1200 packets.

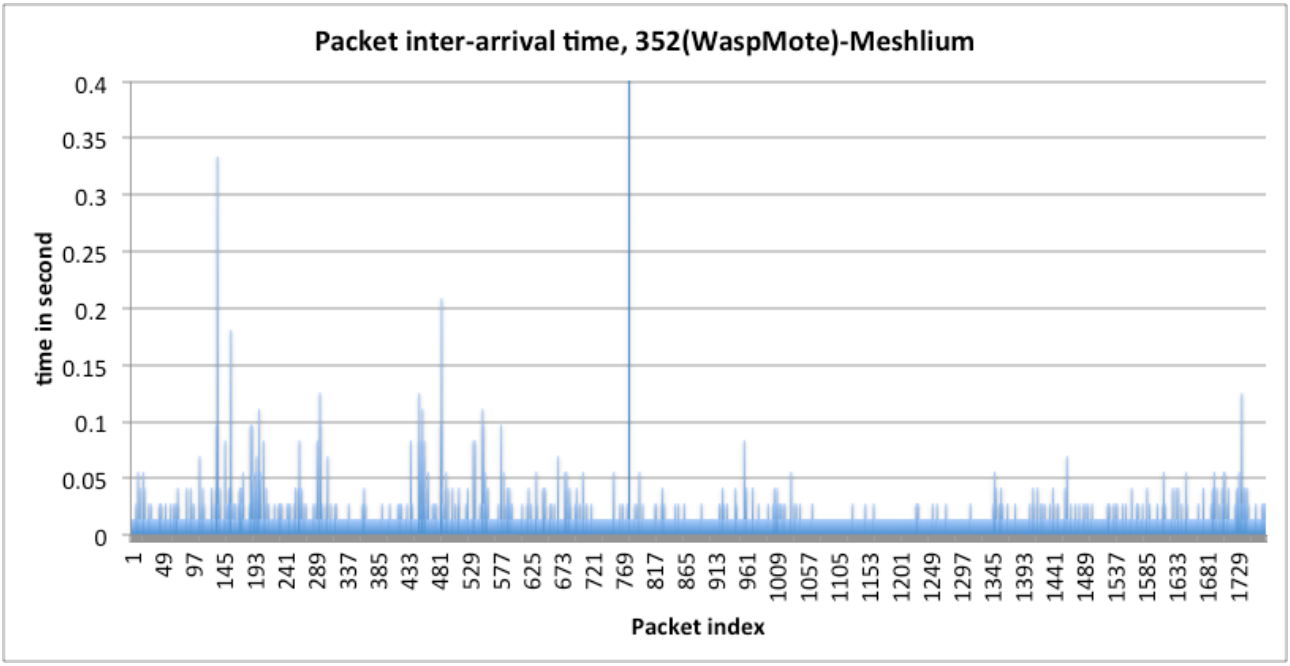


Figure 27 : packet inter-arrival time, test #6

Figure 28 shows for the first ON period, where the number of packet losses is higher, the inter-arrival time after one or more packet losses, in descending order. Without packet losses, the inter-arrival time is around 0.0139s as in test #1.

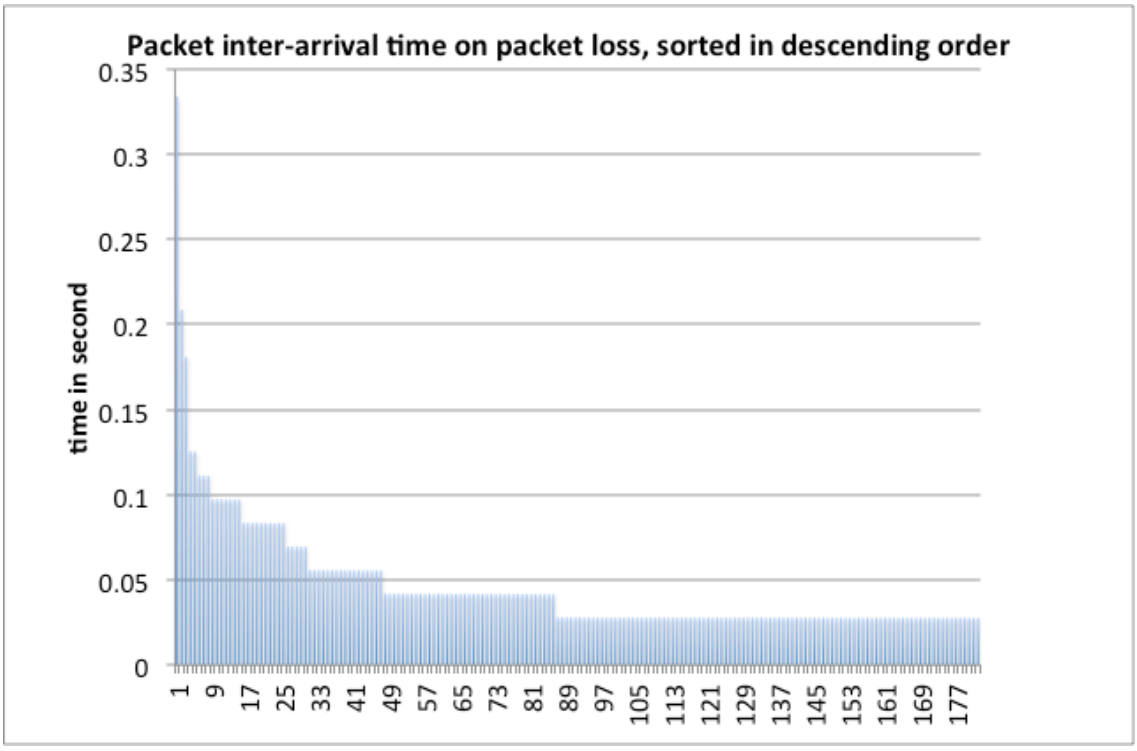


Figure 28 : packet inter-arrival time after losses, test #6. First ON period, descending order.

Packet loss rate in dense, urban, non line-of-sight transmission

Test #7 consists in a non line–of-sight transmission with the audio WaspMote. Figure 29 shows the test location where the source audio WaspMote is placed at location 353, around the corner when compared to the previous test.

Figure 29 : test #7 at location 2

Figure 30 shows the packet inter-arrival time of the ON periods of the audio WaspMote. The line at the middle of the graph is the OFF period. We observed a total of 1494 lost packets out of a total of 2400 packets. The packet loss rate here is therefore 62.25%. We can clearly see here the impact of the non line-of-sight transmission on the packet loss rate in a dense urban area.

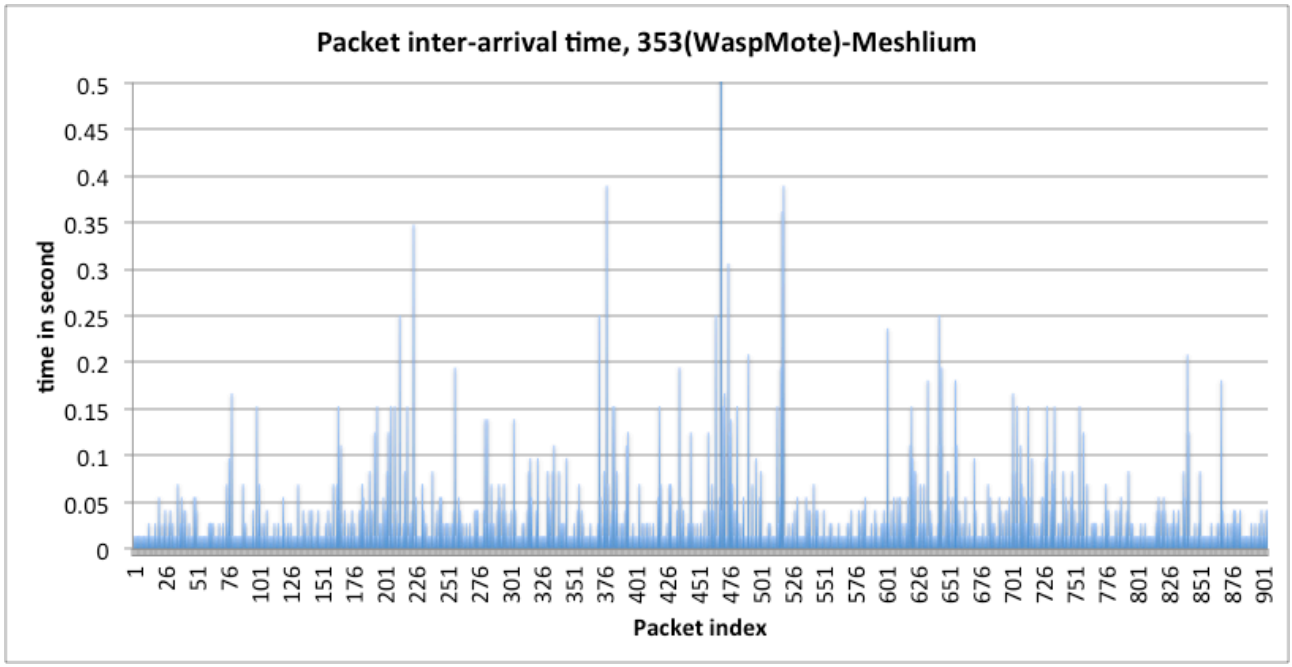


Figure 30 : packet inter-arrival time, test #7

Test #8 now consists in a non line–of-sight transmission with the developed audio board and the TelosB. Similar to the previous case, the source audio mote is placed at location 353. Again, the aggregation level is A6 (giving a 96-byte radio packet).

Figure 31 shows the packet inter-arrival time during a 25s (approximately) audio capture. As can be seen, the packet loss rate is very high (inter-arrival time is very high). We observed 185 lost packets out of a total of 201 (only 16 packets are received), thus giving a packet loss rate of about 92.03%. This is mainly explained by the much weaker transmission power of the CC2420 radio module of the TelosB compared to the one of the XBee module (0dBm against 10dBm).

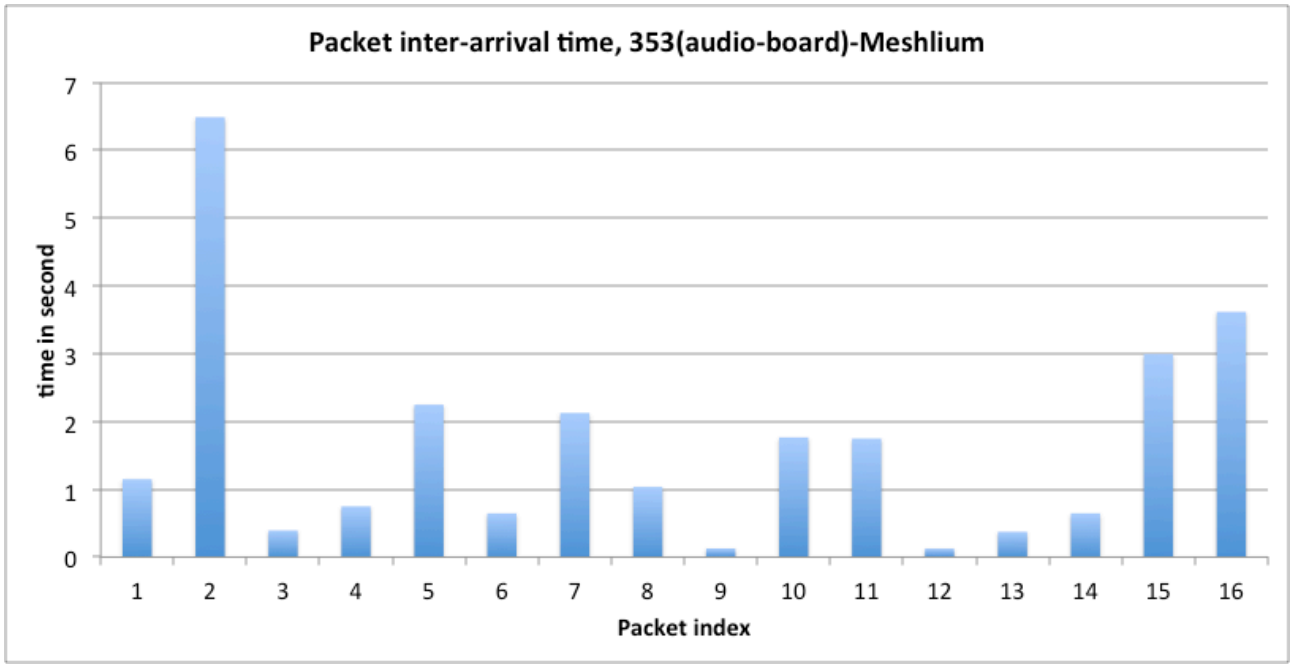


Figure 31 : packet inter-arrival time, A6 level, test #8

Figure 32 shows the wall-clock time on the x-axis at which packets have been received. Normally, the receiver should receive a 96-byte packet every 120ms as shown in figure 25 describing test #3. Here we can clearly see the high number of packet losses.

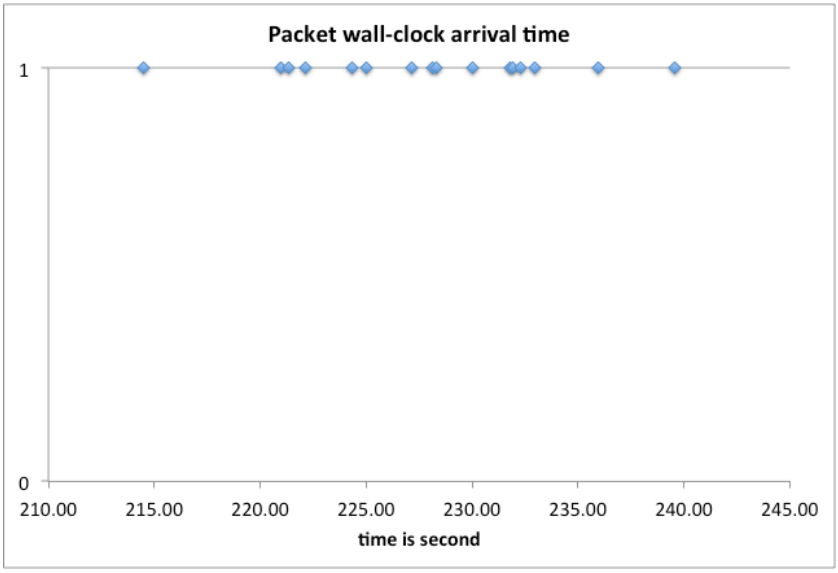


Figure 32 : packet wall-clock arrival time, A6 level, test #8

Test #11 uses location 3 on radio channel 12 with both background traffic from other sensors and many WiFi networks as depicted in figure 33. We again use A6 aggregation level here to be in the same condition than relaying scenario.

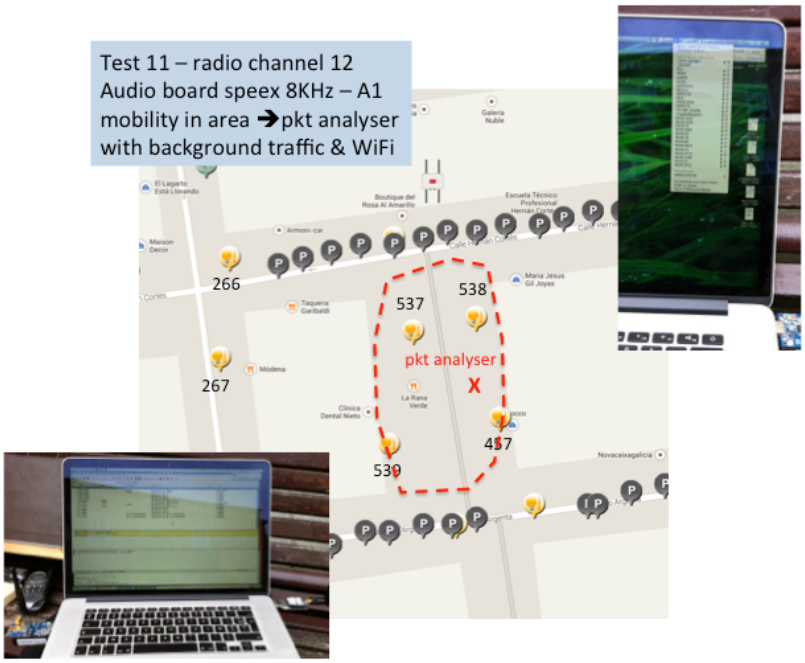


Figure 33 : test #11 at location 3

Figure 34 shows the packet inter-arrival time during a 40s (approximately) audio capture. Here, we observed 2 lost packets out of a total of 324 (giving a packet loss rate of about 0.61%).

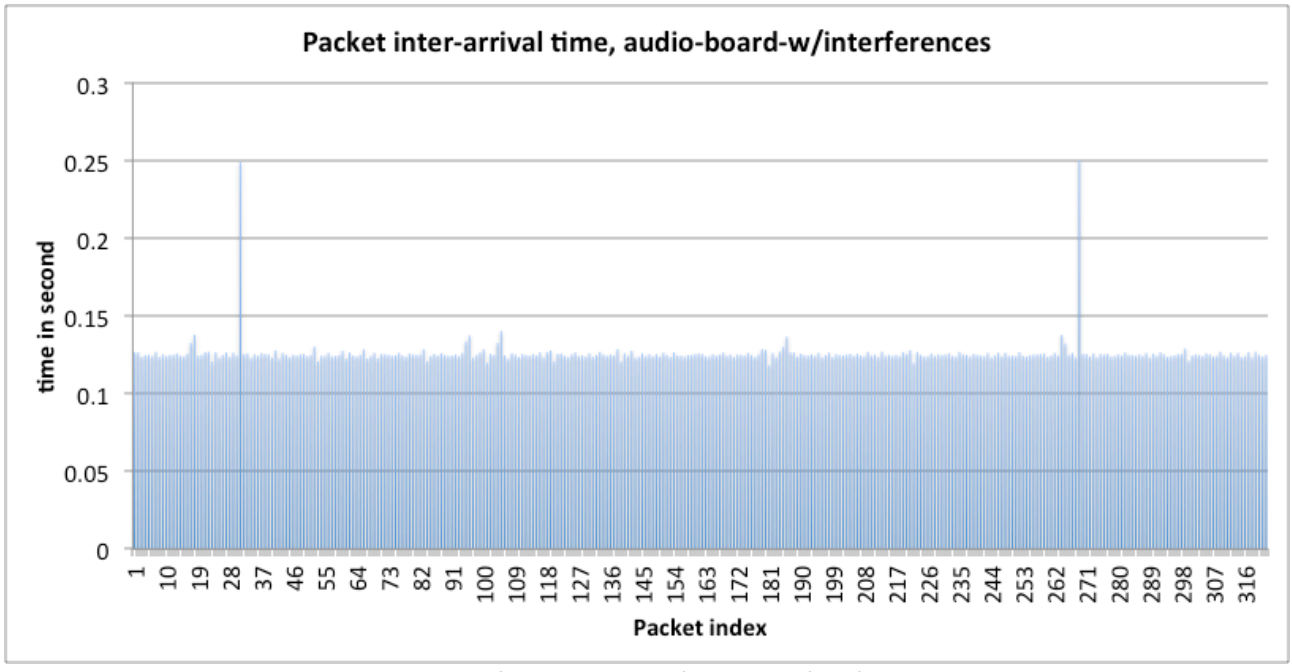


Figure 34 : packet inter-arrival time, A6 level, test #11

If we take the mean then we have a mean inter-arrival time of about 0.1258s with a standard deviation of 0.01. The packet jitter is then, once again, very small. Although we can not be categoric, this test shows that SmartSantander background service traffic and WiFi interferences do not have much impact on the audio traffic.

2-hop transmission: source, relay and destination

Test #4 and **test #5** were performed at Santander's location 1. The Telosb audio board is placed at location 11 and transmits to the Meshlium through a WaspMote relay node placed at location 392. Figure 35 below illustrates the test scenario. Test #4 uses A1 aggregation level while test #5 uses A6.

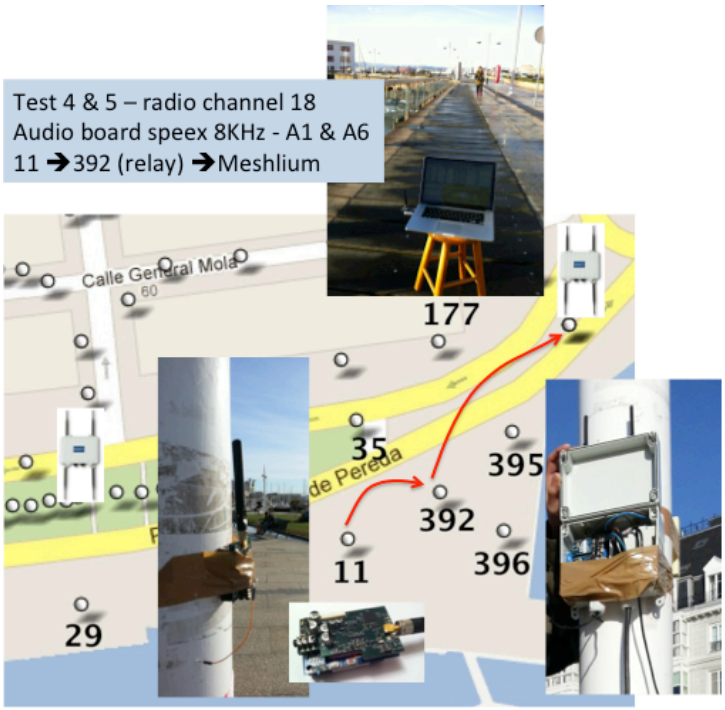


Figure 35 : test #4 and #5 at location 1

For test #4 the packet capture trace of the TelosB audio board shows a total number of packets of 3216 for about 64s of audio capture and transmission. 176 packets were not received, so 3040 were correctly captured by the packet promiscuous sniffer. The packet loss rate is about 5.78%. With no packet losses, the mean inter-arrival time is about 0.02078s with a standard deviation of 0.001 showing that the packet jitter at the source is once again very small.

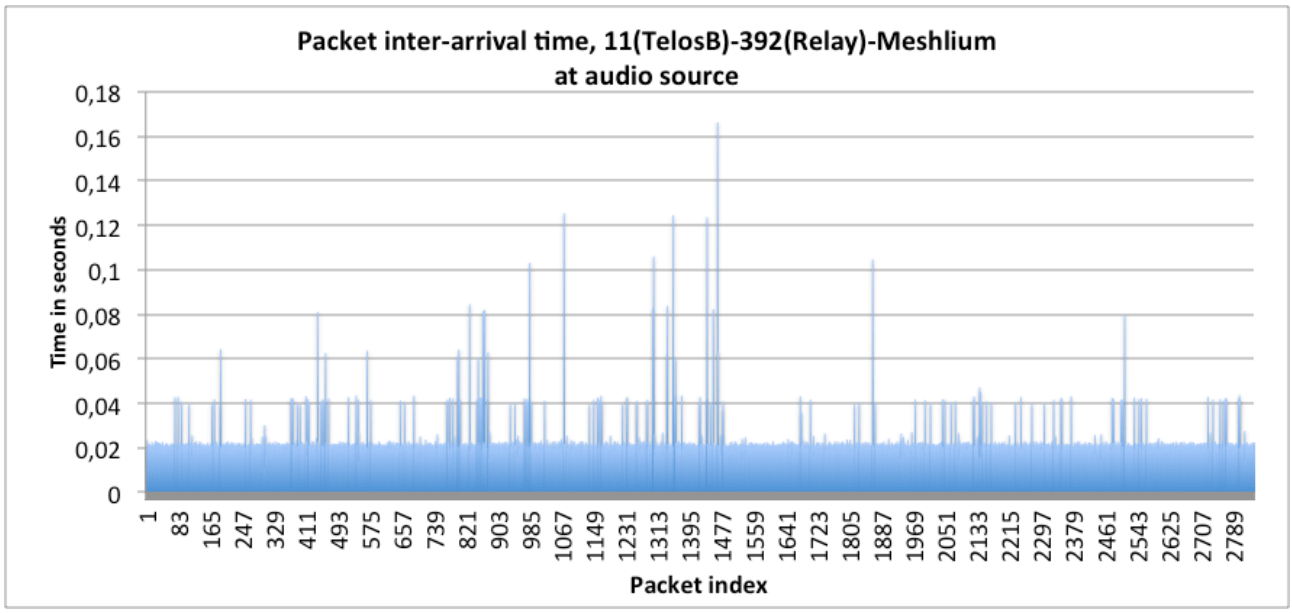


Figure 36 : packet inter-arrival time from the audio source, A1 level, test #4

Now, if we look at what is relayed by the WaspMote relay node in test #4, we observed 817 successfully transmitted packets out of the 3040 packets successfully sent by the audio source and captured by the promiscuous sniffer. This means that only 26.87% of packets has been successfully transmitted (73.13% of packet dropped or lost) by the relay node and received by the Meshlium. With the IEEE 802.15.4 sequence number, we were able to determine that the relay node successfully received at least 883 packets but 66 packets were lost during transmission to the Meshlium. The difference between 3040 and 883 (2157 packets) are packets that were most probably dropped at the relay node due to buffer overflow because of the A1 aggregation level at the audio source: the mean packet inter-arrival time from the relay node (see figure 37) in case of no packet drop is about 0.0609s (see relaying latencies as shown previously in figure 6(top) for a 25-byte packet) while the audio source sends 1 packet every 0.020s.

The 2-hop packet loss rate can be determined by taking 3216 as the initial number of audio packets and 817 as the number of received packet at the Meshlium: 74.6%.

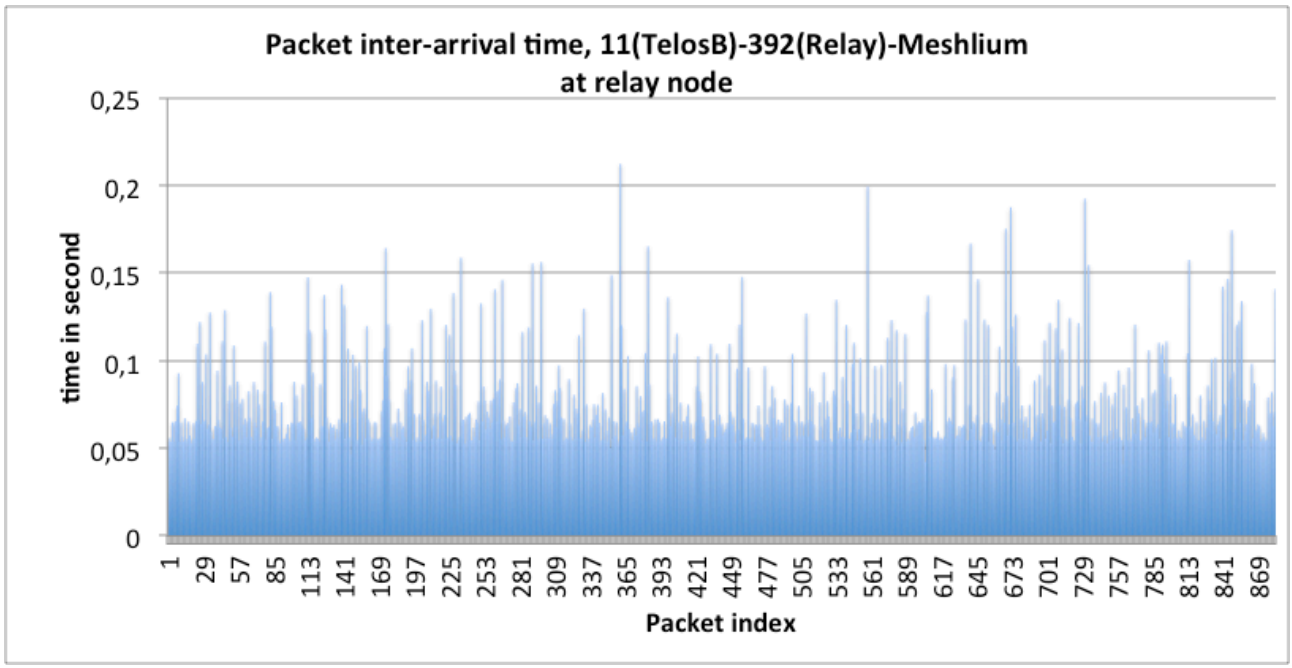


Figure 37 : packet inter-arrival time from the relay node, A1 level, test #4

For **test #5**, the audio source aggregation level is A6, therefore more suitable for WaspMote relaying overhead of about 105ms (96-byte packet). The packet inter-arrival time from the audio source is very similar to what was presented in test #3 with figure 25, therefore we are not reproducing this graph. The important information is that 312 packets were sent for about 38s of audio capture (theoretically we have 312*6*0.02 because of A6 aggregation mode). We observed 43 lost packets so 269 packets are actually received by the relay node (packet loss rate of about 13.78%). Figure 38 shows the packet inter-arrival time from the relay node for test #5.

The relay node received 269 packets from the audio source. We observed 31 lost packets while the relay node is relaying to the Meshlium. Therefore we have a packet loss rate of about 11.52%.

Once again, to obtain the 2-hop packet loss rate, we can take 312 as the initial number of audio packets and 238 as the number of received packet at the Meshlium: 23.7%. Compared to the previous case of A1 aggregation level, we can see that proper usage of aggregation level to meet the relaying capability significantly improved the audio transmission.

We also observed some truncated packets because of the lower reliability of the XBee serial communication with the WaspMote microcontroller. We expect to improve this issue in the future to decrease further the 2-hop packet loss rate.

Under no packet losses, the mean inter-arrival time is 0.109s with a standard deviation of 0.0216.

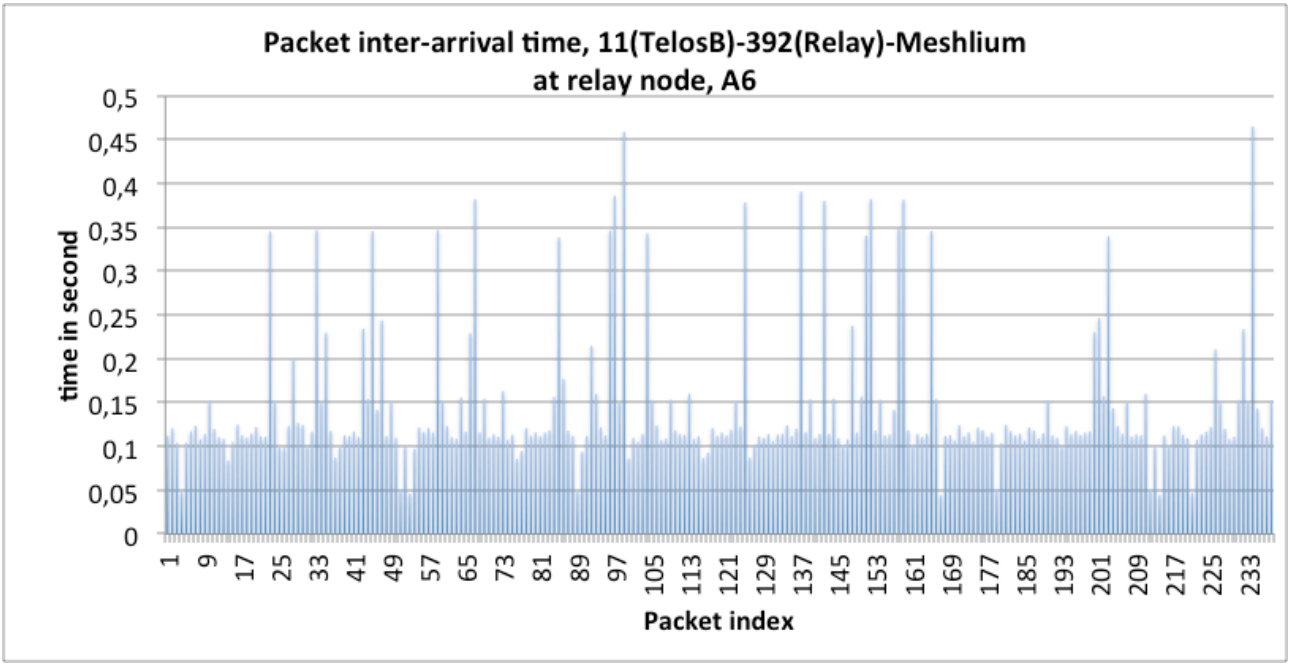


Figure 38 : packet inter-arrival time from the relay node, A6 level, test #5

In **test #9**, we have the same configuration than test #8 but added a relay node at location 351 as depicted in figure 39 below. Audio source aggregation level is A6.

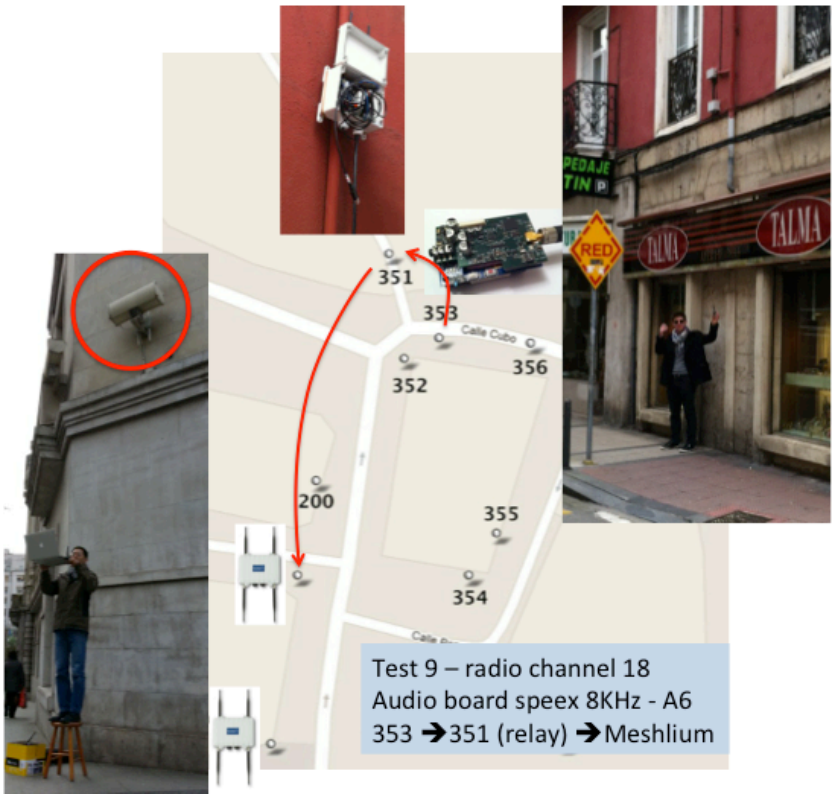


Figure 39 : test #9 at location 2

Figure 40 show the inter-arrival time of all audio packets both from the audio source and from the relay node. However, similar to test #8, packets from the audio board suffer from many losses and only 9 packets were received. They are indicated with red bars in figure 40.

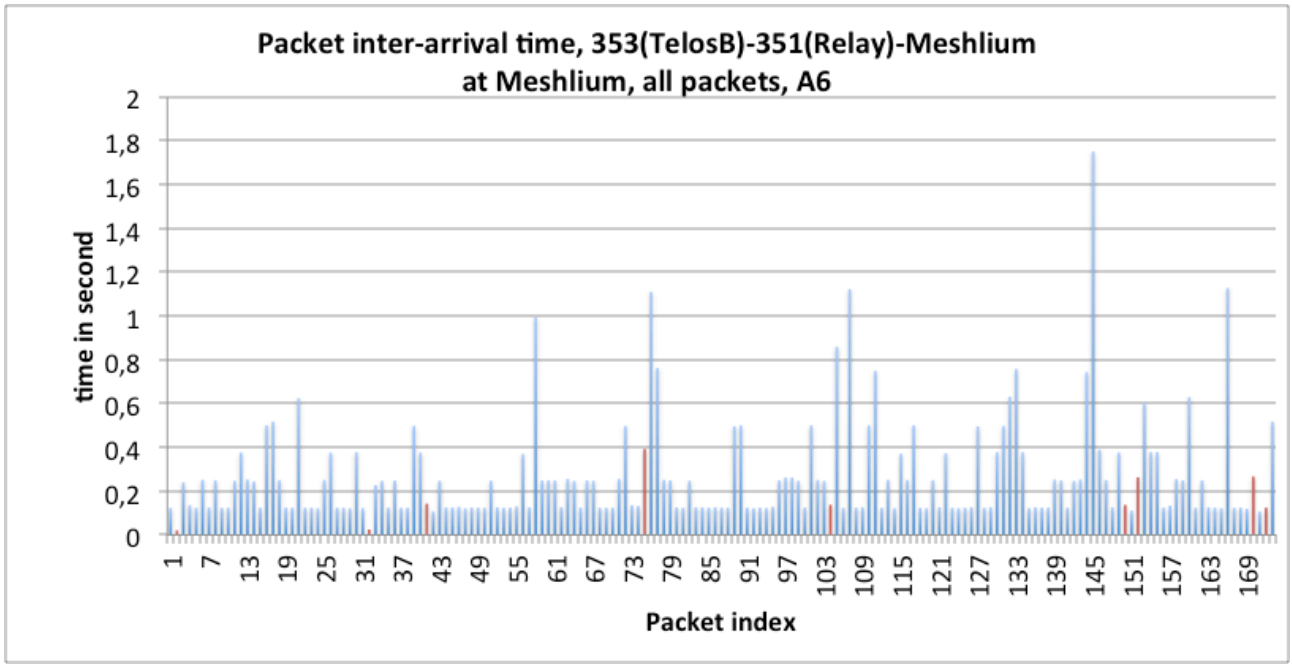


Figure 40 : packet inter-arrival time from both the audio source and the relay node, A6 level, test #9

There have been 352 packets received by the relay node and 137 have been successfully relayed and received at the Meshlium. 188 packets were not received thus the packet loss rate is about 53.40%. This test shows that transmission in a dense environment with many buildings, moving people and cars is very challenging.

In **test #10**, we placed the promiscuous sniffer between the relay node and the Meshlium, at the 'x' location indicated in figure 41 in order to capture packet from both the audio source and the relay node.

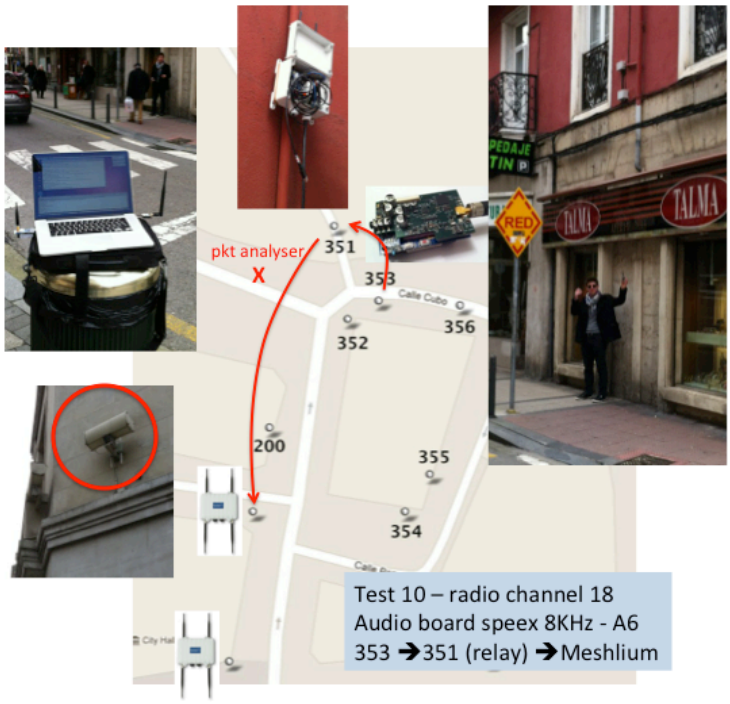


Figure 41 : test #10 at location 2, with the packet analyser in the middle

Figure 42 shows the inter-arrival time from the audio source. We can see that there are very few packet losses (4 packets were lost out of a total of 405 packets representing about 50s of audio capture at aggregation level A6).

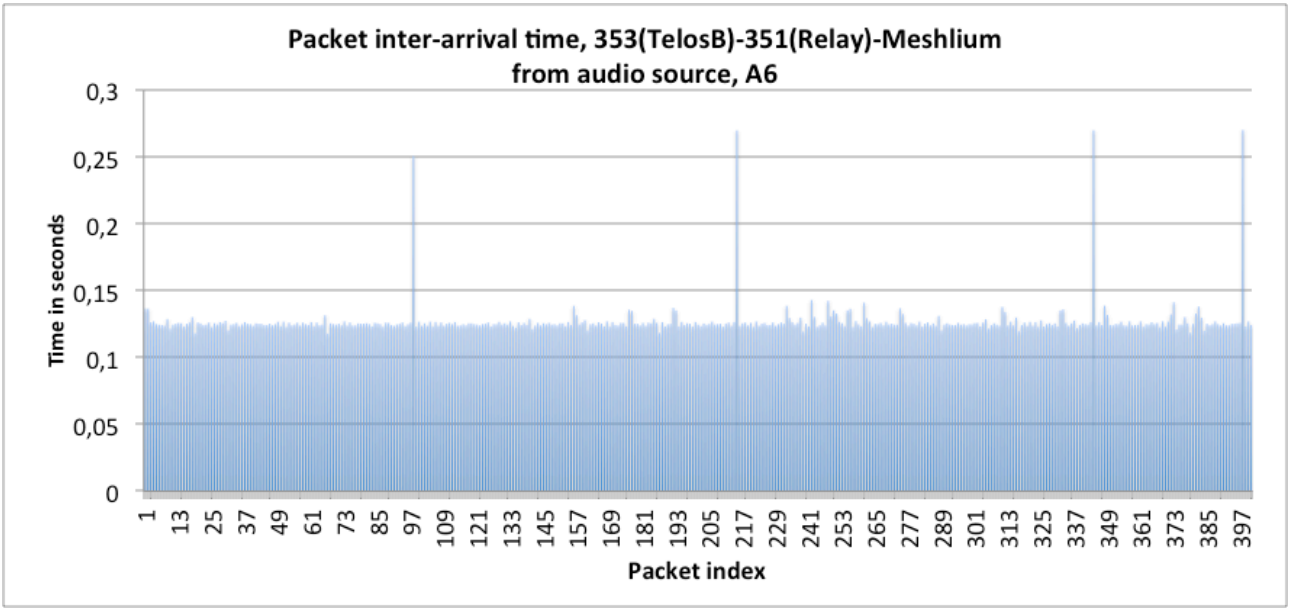


Figure 42 : packet inter-arrival time from the audio source, A6 level, test #10

Figure 43 shows the inter-arrival time from the relay node. The relay node received 400 packets but only 398 were successfully captured by the promiscuous sniffer (2 packets were lost and they are shown in red bars, the 3 other packets with higher inter-arrival time are due to relaying delays). Again, we can see that there are very few packet losses.

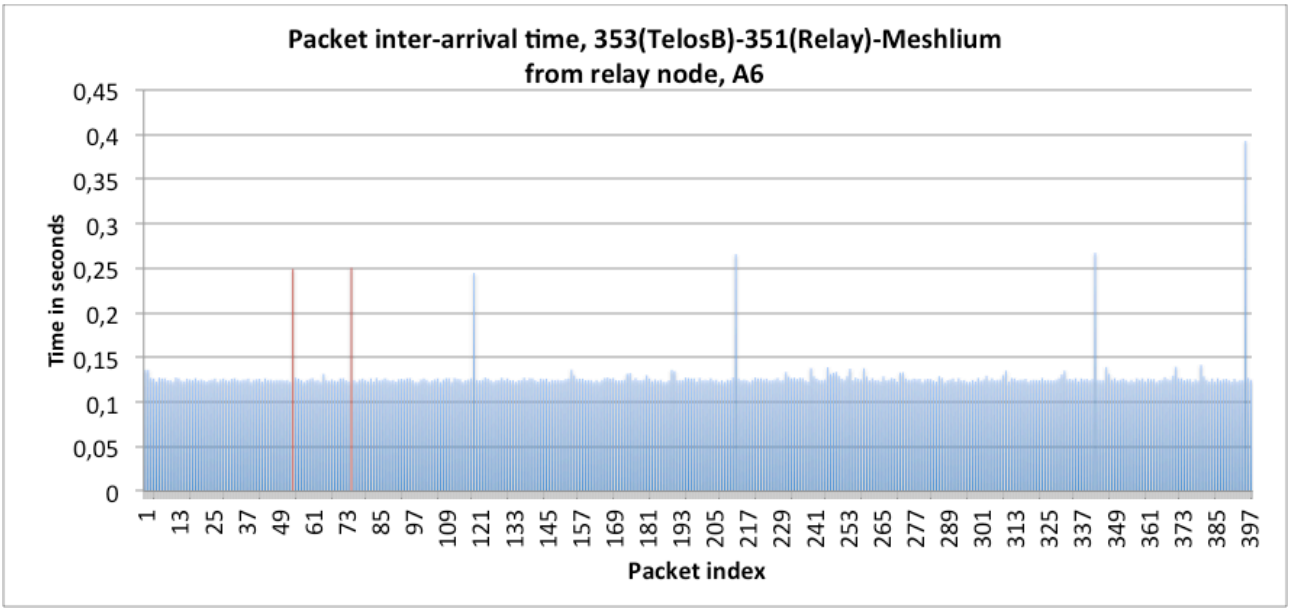


Figure 43 : packet inter-arrival time from the relay node, A6 level, test #10

The results of test #10 actually confirm that the high proportion of packet losses of test #9, where the promiscuous sniffer was placed at the Meshlium, is mainly due to the relay node-Meshlium link (from location 351 to Meshlium). Relay node placement or selection then have a very strong impact on relaying reliability as most nodes are placed against building walls.

Test #10 also provides measures of the packet relaying time at the WaspMote relay node. The next table reproduces the first lines of the wireshark capture where we can see in the "delta time" column the difference between the time at which the packet from the audio source is

captured and the time at packet from the relay node has been captured. This time can be considered as the relay time. The audio source has address 0x0090 and the relay node has address 0x1ddf.

index	time	src	dest	type	SN	delta time	data
4	885,225984	0x0090	0x1ddf	Data,	25	0,664384	FF55C9141DD8C9500039CE702040AFAC7A62B310ED2DDA1B...
5	885,331936	0x1ddf	0x0100	Data,	32	0,105952	FF55C9141DD8C9500039CE702040AFAC7A62B310ED2DDA1B...
6	885,362464	0x0090	0x1ddf	Data,	26	0,030528	FF55CF143EA7AB5B5F12B580C00957F05C1020E81DD5046C...
7	885,467968	0x1ddf	0x0100	Data,	33	0,105504	FF55CF143EA7AB5B5F12B580C00957F05C1020E81DD5046C...
8	885,48864	0x0090	0x1ddf	Data,	27	0,020672	FF55D5141DC1C4767E82A1C8E116968030A08B7F9FD48DC1...
9	885,595072	0x1ddf	0x0100	Data,	34	0,106432	FF55D5141DC1C4767E82A1C8E116968030A08B7F9FD48DC1...
10	885,615552	0x0090	0x1ddf	Data,	28	0,02048	FF55DB141DD8C47C40ADBE59D040A1A3D8A09BD646303DD7...
11	885,721312	0x1ddf	0x0100	Data,	35	0,10576	FF55DB141DD8C47C40ADBE59D040A1A3D8A09BD646303DD7...
12	885,740352	0x0090	0x1ddf	Data,	29	0,01904	FF55E1141B955E93BE9EA772A113FB031260EBE97F58BFFB...

We can see that this relay time is about 105ms is these few lines. Figure 44 plots the relay time of all relayed packets (398 packets) and the mean value is 0.10854 with a standard deviation of 0.00255. This is quite consistent with what was previously measured for WaspMote motes and shown in figure 4(top).

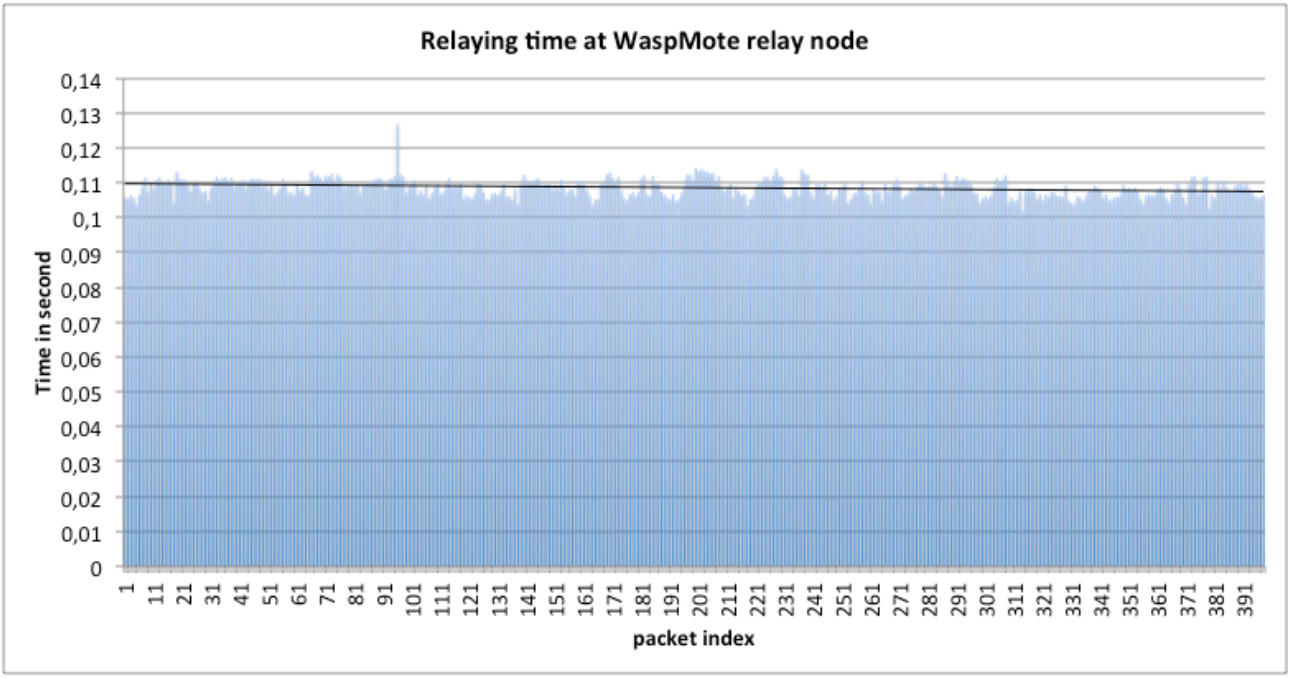


Figure 44 : relay time of the relay node, A6 level so 96-byte packet, test #10

Conclusion of benchmark tests in Santander's SmartSantander test-bed

We summarize the main results of the benchmark tests performed in Santander in the table below:

Santander, SmartSantander test-bed			
test scenario	Pkt jitter at source	pkt jitter at relay	pkt loss rate
1-hop LOS open space	very small (1)	NA	0% - 12% (2)
1-hop LOS urban	very small (1)	NA	35% (2)
1-hop NLOS urban	very small (1)	NA	60% - 92% (2)
2-hop open space	very small (1)	very small (4)(5)	5% - 23% (3)
2-hop urban	very small (1)	very small (5)(5)	53% (3)

1. The packet jitter at the source, for both the WaspMote audio mote and the AdvanticSys TelosB audio board, is very small and can be easily compensated at the destination with a very simple playout buffer.

2. The packet loss rate at 1-hop in LOS condition, and when the distance of next hop is similar to what can be found in Santander, is very small. In non-LOS condition, for instance with buildings in-between, the packet loss rate can be very high: we for instance found packet loss rate as high as 92% with the developed audio board in a dense urban environment in non-LOS condition.

3. At 2-hop or more, using relay nodes, non-LOS condition can be overcome, and reception quality can greatly be improved. This is particularly important in in-door environment as shown in the HEPIA building. However, the choice of the relay nodes can have a big impact of the performances. In urban environment, the packet loss rate can still be high and more hops may be needed at the cost of higher latencies.

4. The packet relaying times measured with a promiscuous sniffer are consistent of what have been predicted in the previous deliverable. According to the maximum relaying capabilities, an appropriate aggregation level at the source can be used to reduce the packet losses at intermediate relay nodes.

5. The packet relay jitter was found again quite small and can be easily compensated at the destination with a very simple playout buffer.

Therefore, as a result of the benchmark tests, the SmartSantander test-bed in Santander is capable of supporting streamed audio both in open space and urban environment when LOS transmission is possible. in NLOS conditions, 1-hop transmission is not capable of providing a sufficiently small packet loss rate for an acceptable audio quality (packet loss rate much higher than 35%). Using 2-hop or more transmission can leverage the NLOS conditions and decrease the packet loss rate in urban environment. However, the choice of the relay nodes is of critical importance increase transmission quality and there are certainly many interesting issues to dynamically chose the right relay nodes. In all cases, the packet jitter at the source and at the relay nodes is very small.

4.2 Tests in Geneva (HobNet, HEPIA site)

The second set of tests is performed at HEPIA site in Geneva. We chose this site because it is quite representative of various environments that can be found in buildings for Smart Buildings purposes. Figure 45 shows various parts of HEPIA building with long corridors (3), student/public restaurants and halls (1) and even an in-door chimney (2) with quite interesting transmission particularities.

Figure 45 : various images of the HEPIA building

In total, we performed 7 tests on 3 locations of the HEPIA building:

- Location 1: test #1, #2 and #3
- Location 2: test #4
- Location 3: test #5, #6 and #7

The tests are also divided into 1-hop and 2-hop transmission:

- 1-hop: test #1, #4, #5, #6
- 2-hop: test #2, #3, #7

Only AdvanticSys TelosB motes will be used, both for audio source (the developed audio board) and the relay nodes because these are the hardware platforms deployed in HobNet for Smart Buildings applications.

1-hop, source to destination

In **test#1**, the audio source is placed in location 1 of HEPIA, in the student cafeteria. The audio source mote is strapped on one of the pillar of the cafeteria, somewhere in equal distance from the 2 entrances of the cafeteria. The promiscuous sniffer is moved from one entrance to the other by the outside hall. The configuration of test#1 is illustrated in figure 46 below, with A1 aggregation level, i.e. one 24-byte packet every 20ms.

Figure 46 : test#1 in the main hall of the HEPIA building

Figure 47 shows the packet inter-arrival time during a 86s (approximately) audio capture. As the receiver is moved around the main hall, through the glass wall, we can observe variations on the number of packet losses. In total, we observed 687 lost packets out of a total of 4280 (giving a packet loss rate of about 16%).

The maximum number of consecutive lost packets is 8. The mean inter-arrival time is 0.24

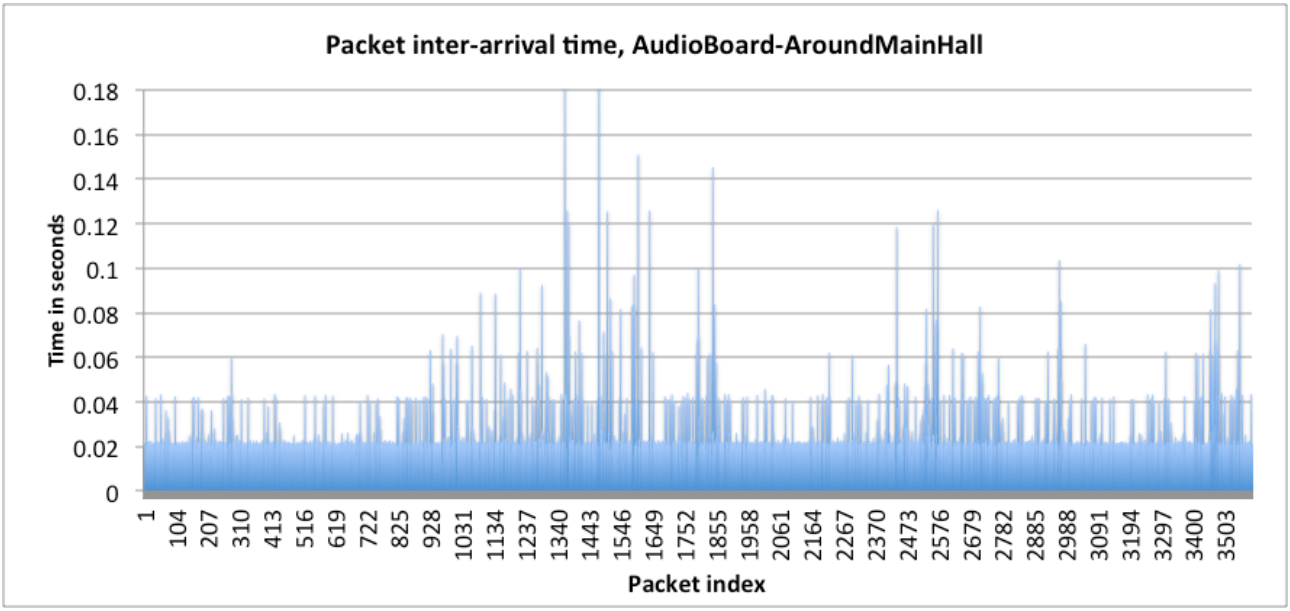


Figure 47 : packet inter-arrival time from the audio board, A1 level, test#1

In **test#4**, we tested the transmission quality on the in-door chimney, see figure 48 below. The audio board is placed on the metallic structure shown with the red rectangle.

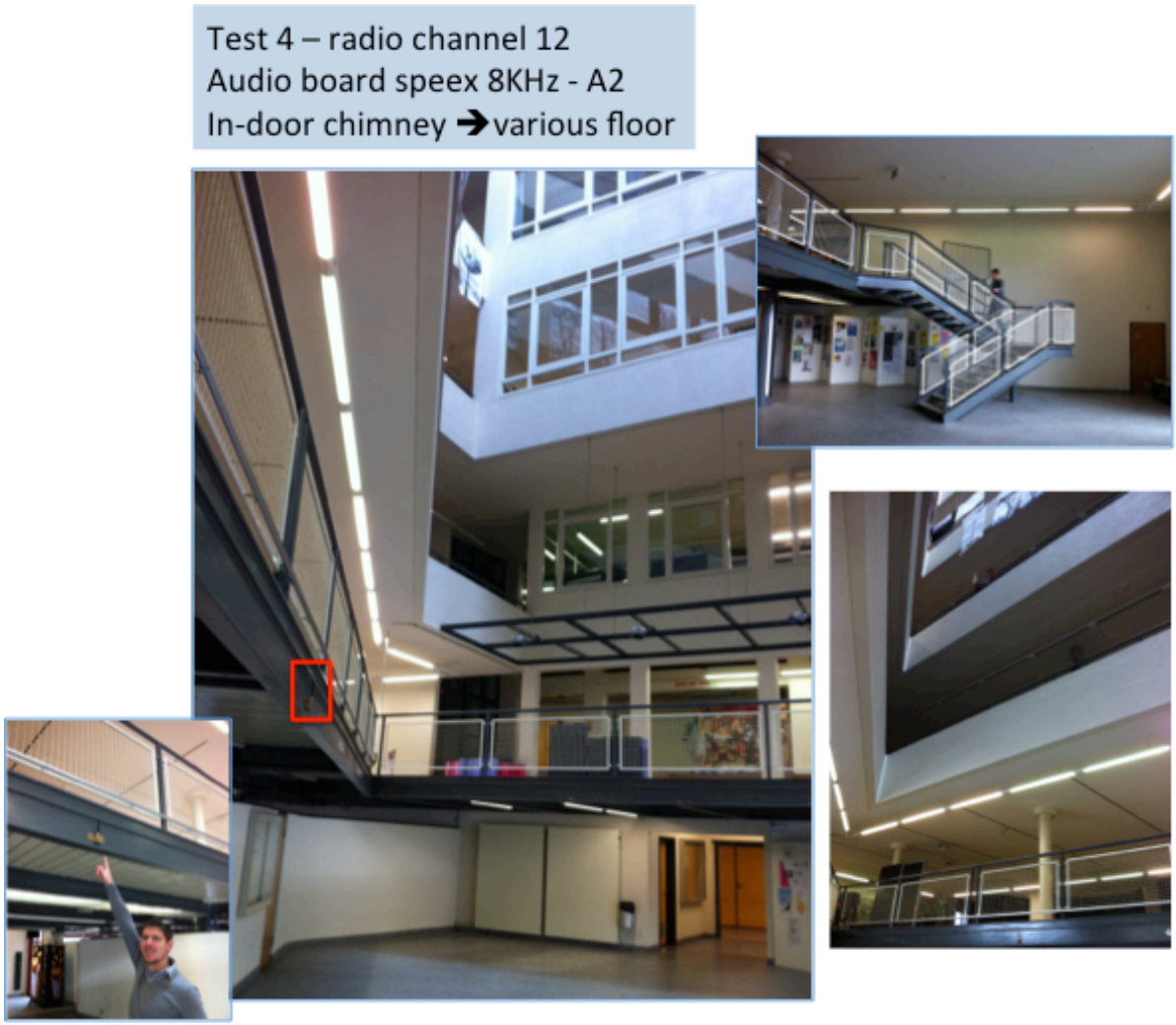


Figure 48 : test#4 in in-door chimney of the HEPIA building

Figure 49 shows the inter-arrival time at the receiver when it moves from the base floor up to the last floor of the building, following the stairs around the chimney.

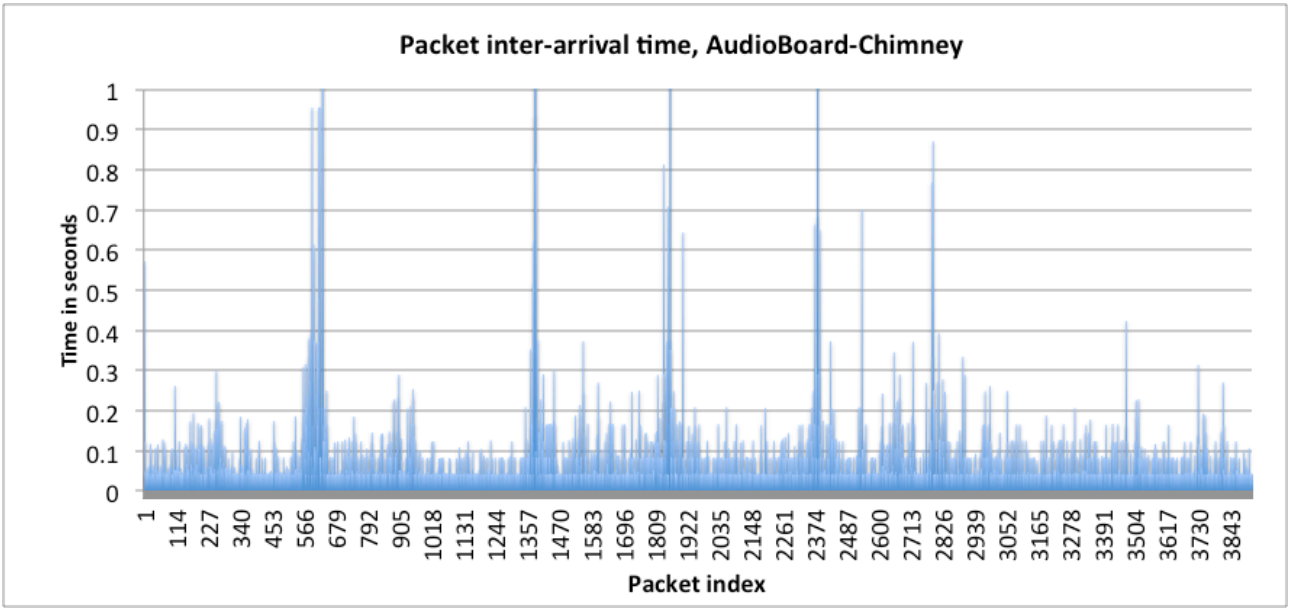


Figure 49 : packet inter-arrival time from the audio board, A2 level, test#4

We observed 2556 packet losses out of a total of 6572 packets. The packet loss rate is therefore quite high, 38.89%.

Figure 50 shows the number of lost packets when the receiver moves. At several moments, we can have 165 lost packets in a row.

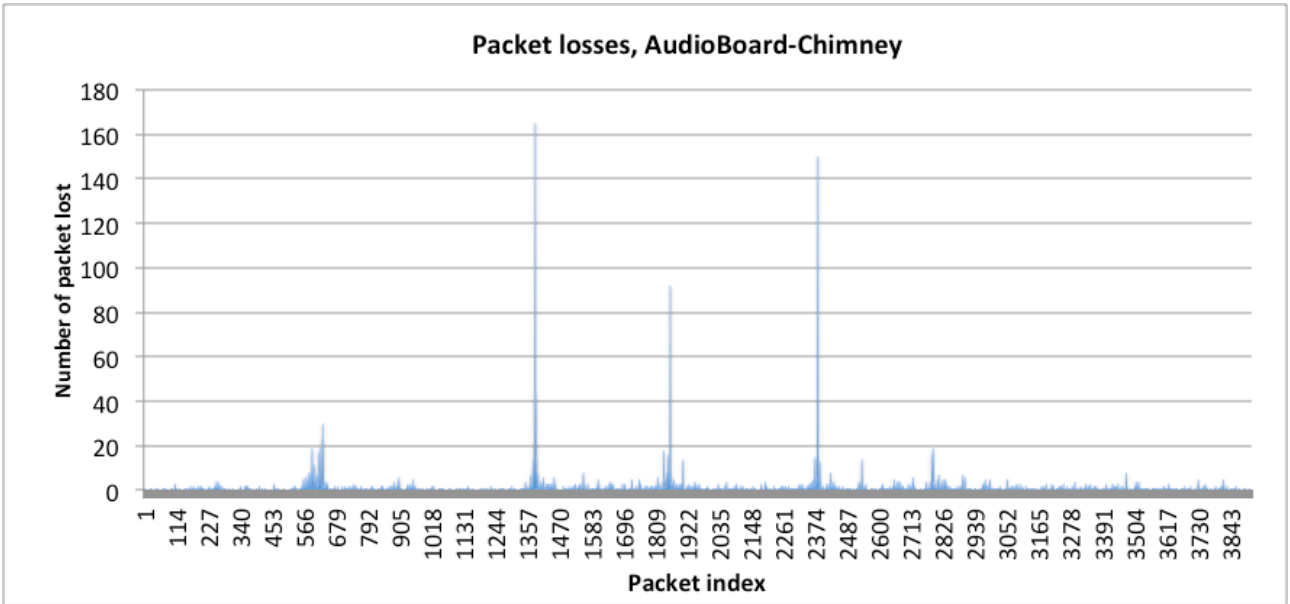


Figure 50 : number of lost packets in a row, A2 level, test#4

In **test#5**, we tested the transmission quality in a long corridor, illustrated by figure 51. The audio board is placed on a concrete pillar at one end of the corridor. We moved the receiver to the other end (to the corridor entrance, towards the stairway), then went down one floor.

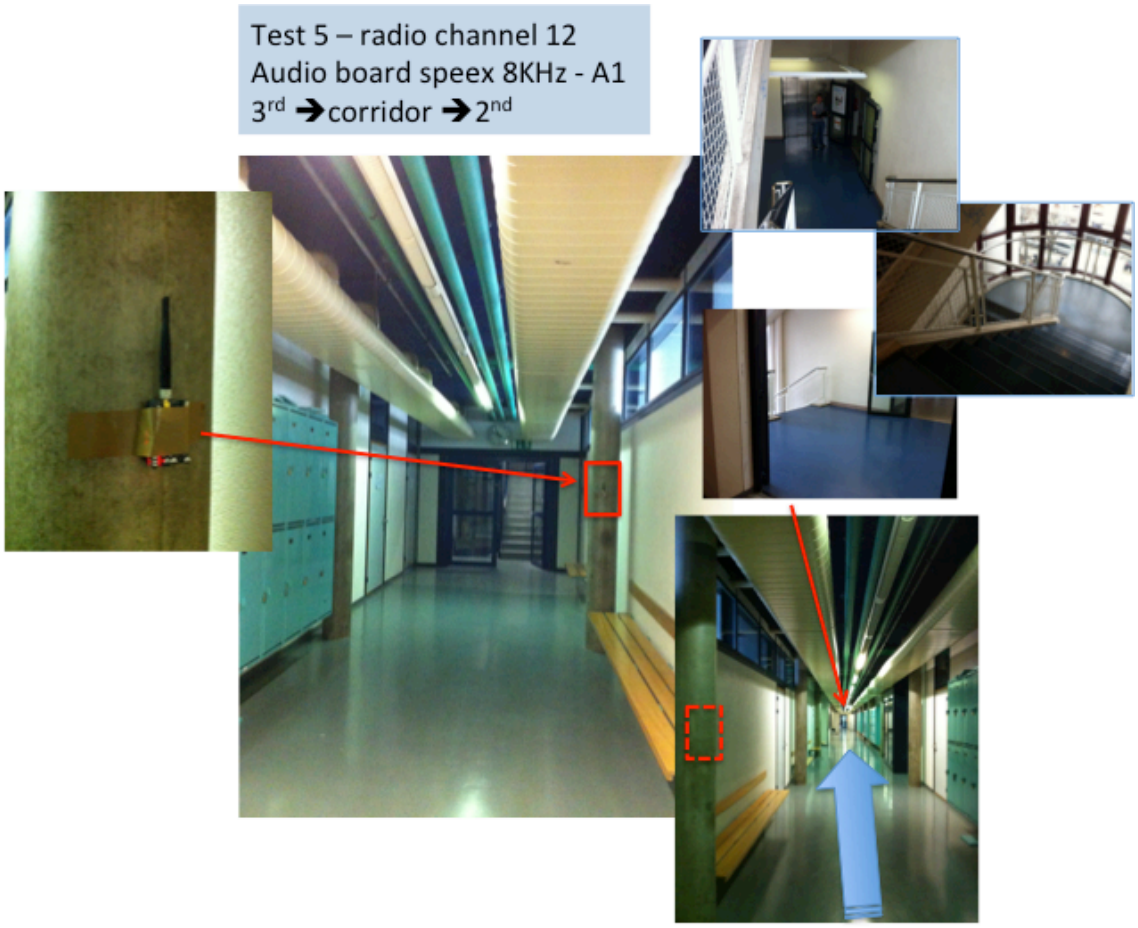


Figure 51 : test#5 in a long corridor of the HEPIA building

Figure 52 shows the number of lost packets as the receiver is moved towards the corridor entrance, to the stairways and to level 2 of the building. We measured 2861 lost packets out of a total of 6713 packets, giving a packet loss rate of about 42.61%.

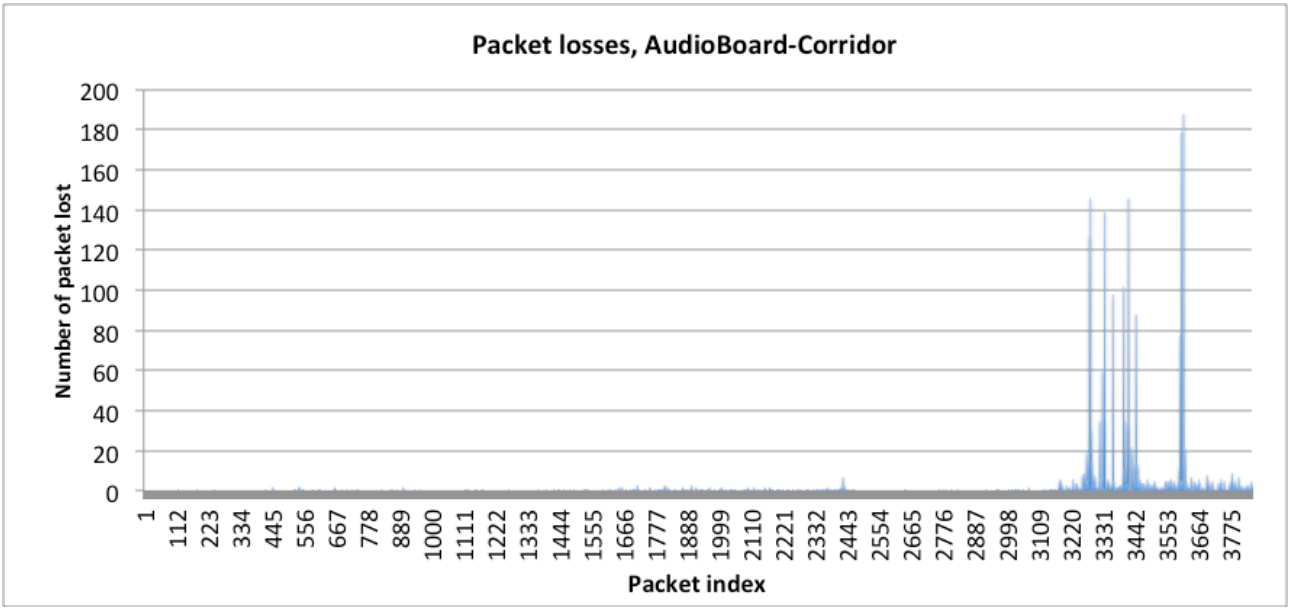


Figure 52 : number of lost packets in a row, A1 level, test#5

We can however observe that the lost packets are concentrated in the right-most part of figure 52, when the receiver was actually in the stairway, towards 2nd floor. Before the receiver went to 2nd floor, the packet loss rate was below 8%.

In **test#6**, we placed the receiver one floor below the audio board as illustrated by figure 54, first near the elevator, then a bit farther. Figure 55 shows the number of lost packets. The packet loss rate is very high, about 81%, especially when the receiver is moved away from the stairs (right-most part of figure 61).

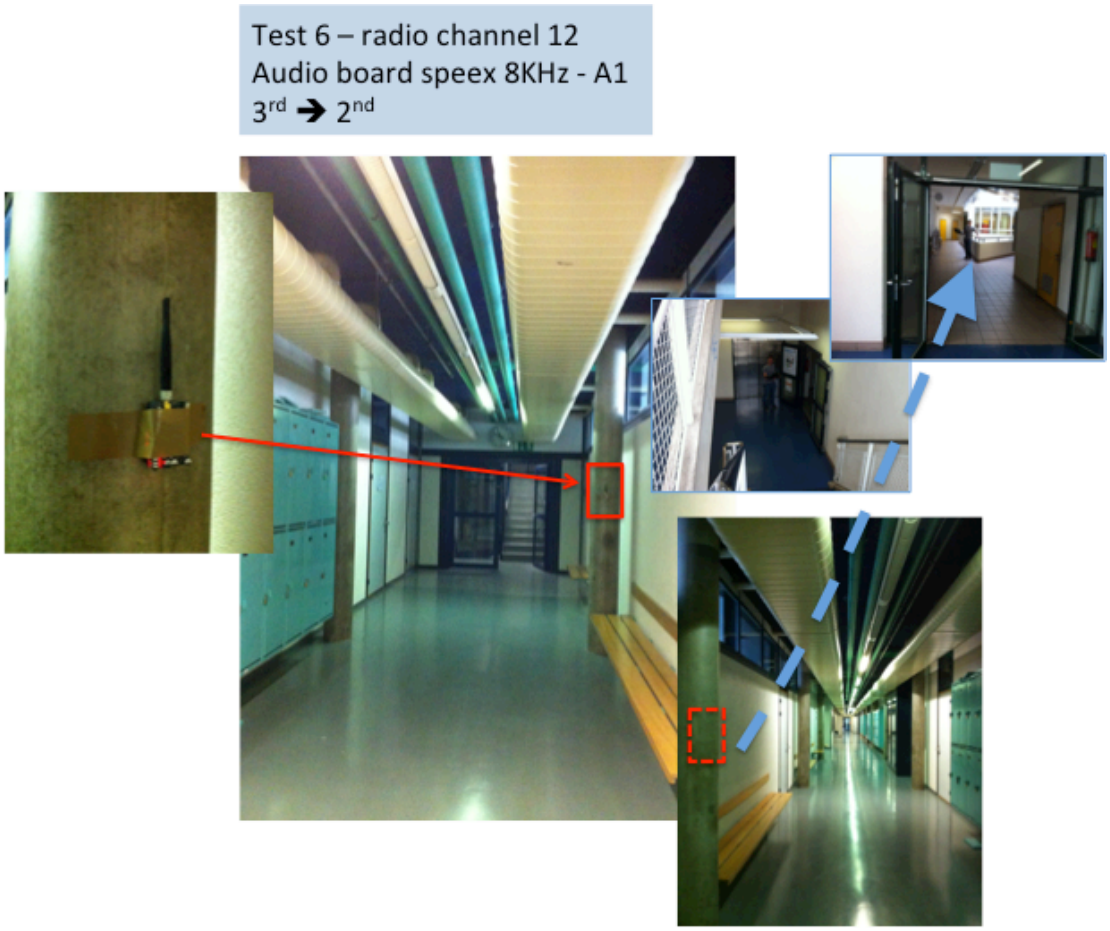


Figure 54 : test#6 in a long corridor of the HEPIA building, receiver in 2nd floor

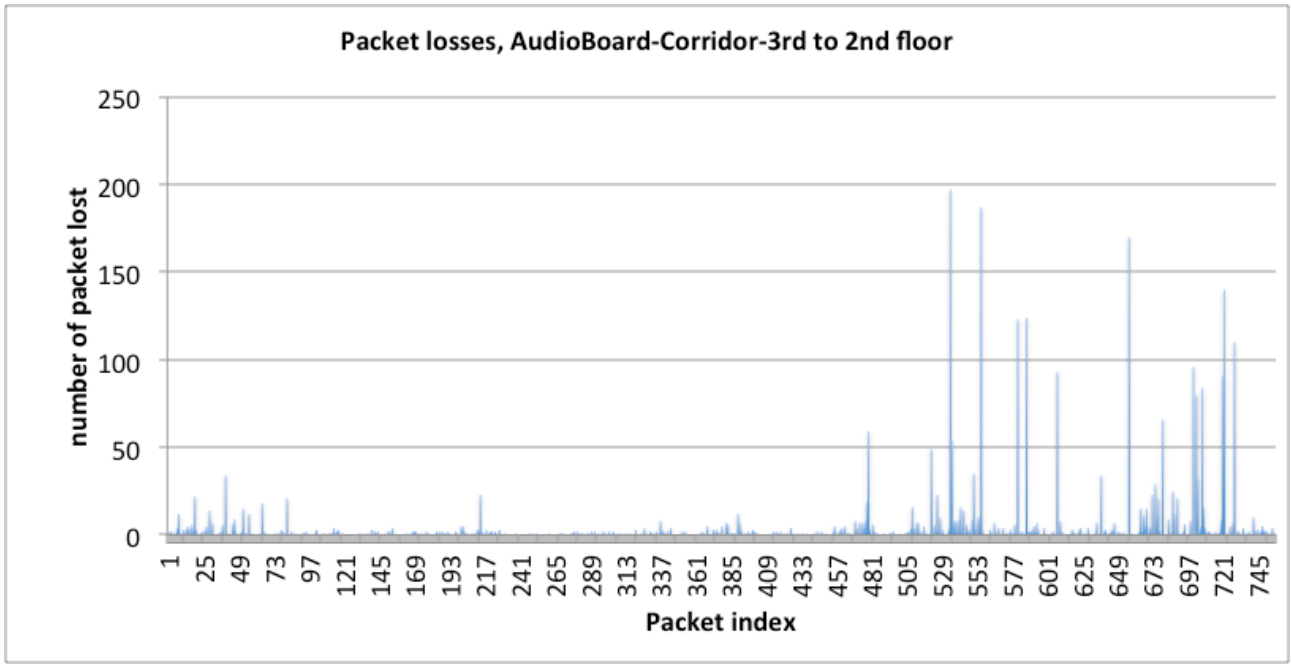


Figure 55 : number of lost packets in a row, A1 level, test#6

2-hop transmission: source, relay and destination

In **test#2** and **test#3** we placed a relay node in the restaurant as shown in figure 56. The audio board is set like in test#1, the relay node is placed between the audio board and the exit in the back of the central picture in figure 56. Test#2 uses A1 aggregation and test#3 uses A2 aggregation.

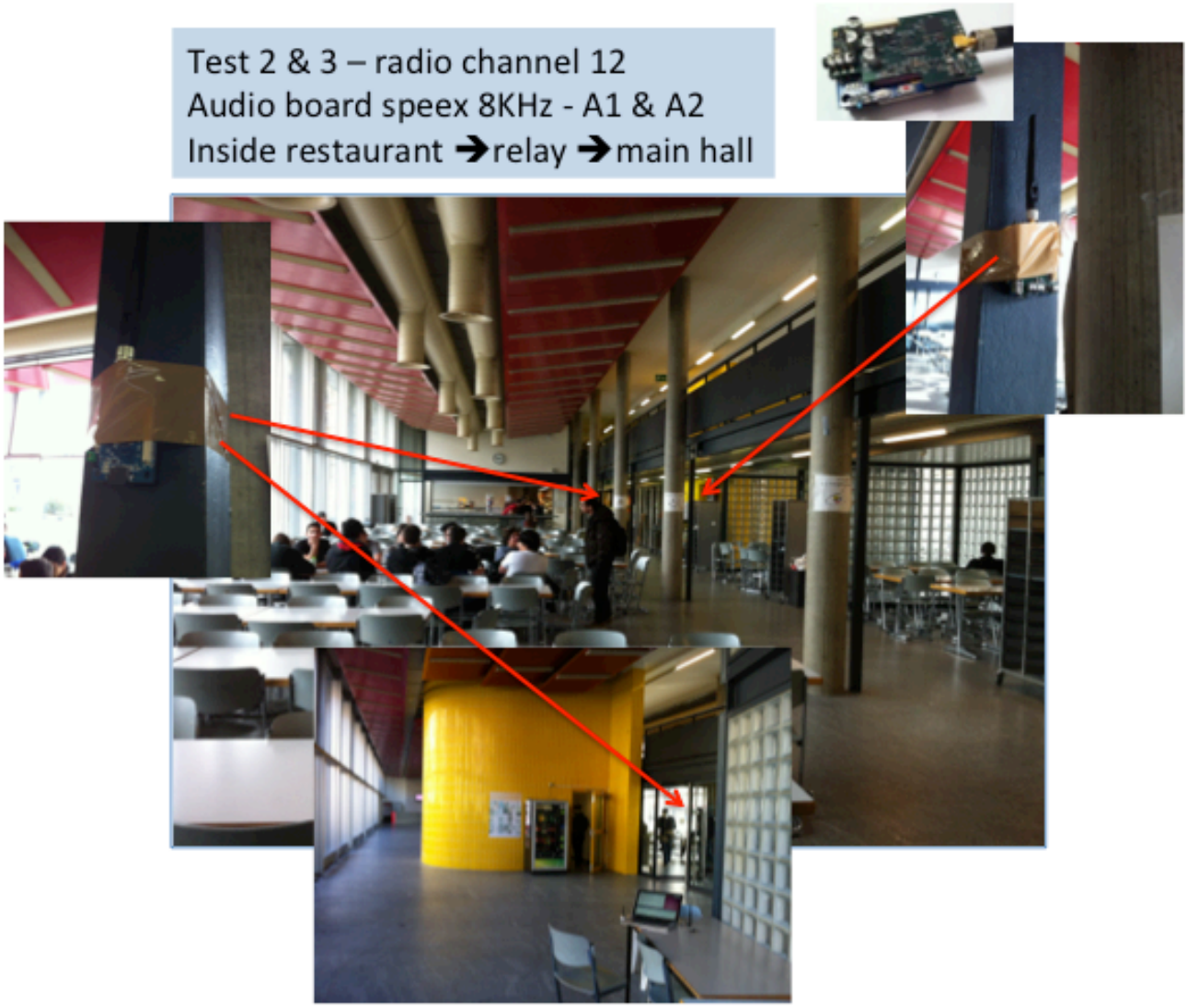


Figure 56 : test#2 & test#3 in the main hall of the HEPIA building, with relay node

Figure 57 shows the number of lost packets. We have observed 86 lost packets out of a total of 1681 packets, resulting in a packet loss rate of about 5.11%.

If we look back at test#1 that did not use the relay node, the right-most part of figure 47 from packet index 3822 to 4280 (458 packets) corresponds to when the receiver was located at the same place (near the exit door) than in test#2. We then observed 62 lost packets, resulting in a packet loss rate of about 13.53%. We can clearly see the benefit of using the relay node inside the restaurant space to improve the reception quality in the main hall.

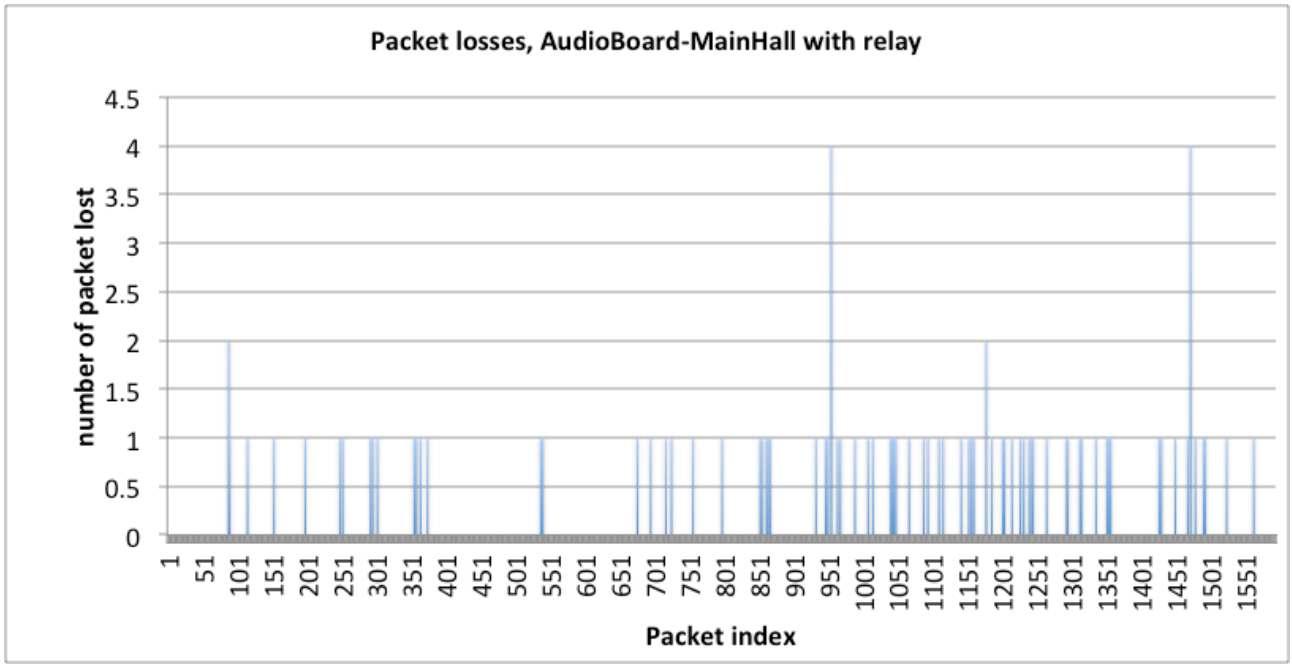


Figure 57 : number of lost packets in a row, A1 level, test#2

We show in figure 58 the output of test#3 with the A2 aggregation level. We observed the same level of packet losses: 30 lost packets out of a total of 520 resulting in a 5.76% packet loss rate.

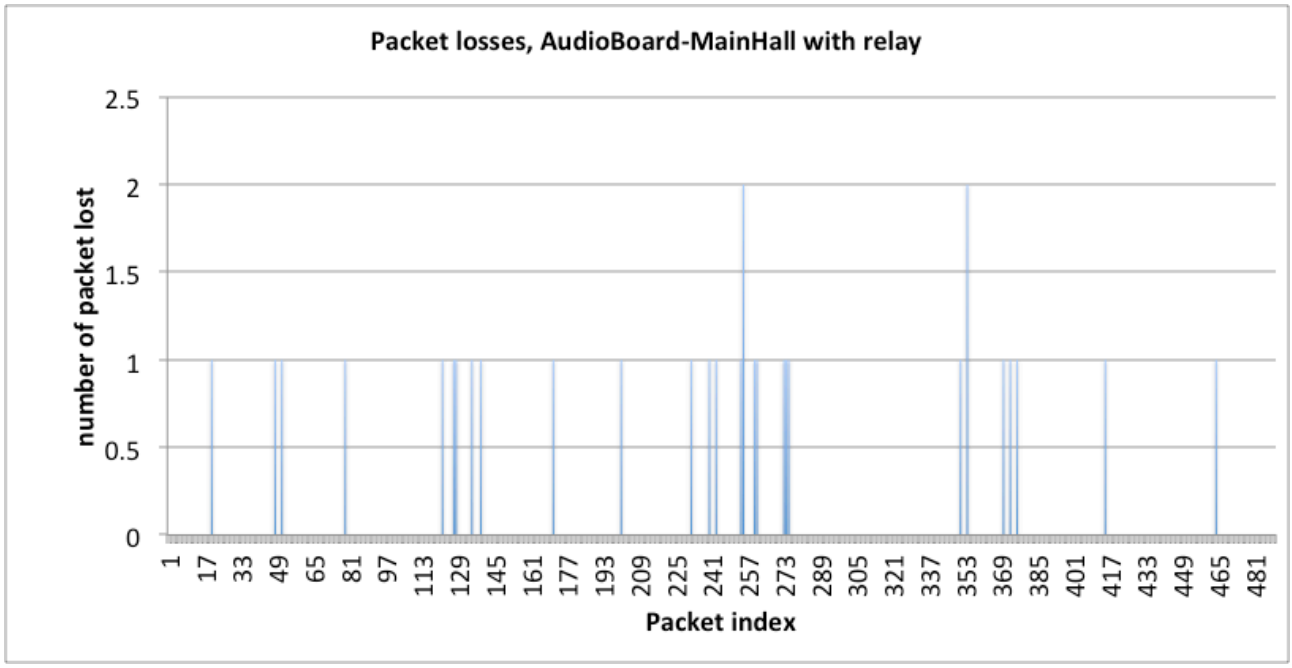


Figure 58 : number of lost packets in a row, A2 level, test#3

In **test#7**, we added a relay node to test#6 where we placed the receiver one floor below (level 2) the audio board, a bit away from the stairways. The relay node was fixed at the corridor entrance, near the stairs to level 2 as shown in figure 59.

Figure 59 : test#7 in a long corridor of the HEPIA building, receiver in 2nd floor, relay at corridor entrance, near the stairways

Compared to test#6 where the packet loss rate was about 81%, we observed here 342 lost packets out of a total of 1170 packets, resulting in a packet loss rate of about 30%. Adding the relay node at the corridor entrance, near the stairways greatly improves the reception quality at one floor below, keeping the packet loss rate below 35% therefore allowing for a reasonable audio quality. Figure 60 shows the number of lost packets for this test.

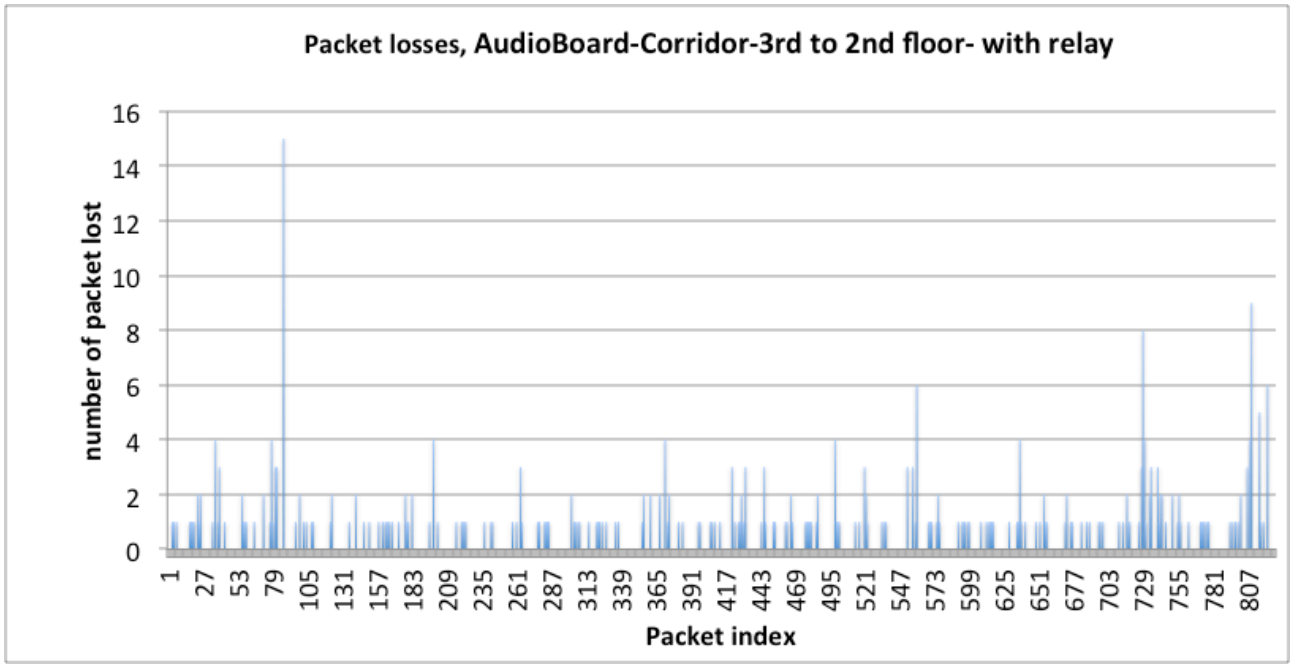


Figure 60 : number of lost packets in a row, A2 level, test#7

Test#7 also provides measures of the packet relaying time at the AdvanticSys TelosB relay node. The next table reproduces the first lines of the wireshark capture where we can see in the "delta time" column the difference between the time at which the packet from the audio source is captured and the time the packet from the relay node has been captured. This time can be considered as the relay time. The audio source has address 0x0090 and the relay node has address 0x0200.

index	time	src	dest	type	SN	delta time	data
1	129.959456	0x0090	0x0200	Data,	89	0.033504	FF55DF141B992B0A54E519CFA180BCE58459739C04E739CE...
2	129.970112	0x0200	0x0100	Data,	209	0.010656	FF55DF141B992B0A54E519CFA180BCE58459739C04E739CE...
3	130.002912	0x0090	0x0200	Data,	90	0.0328	FF55E11419D9A4A4038008402DC007240BB6CB925405AA6A...
4	130.014208	0x0200	0x0100	Data,	210	0.011296	FF55E11419D9A4A4038008402DC007240BB6CB925405AA6A...
5	130.044128	0x0090	0x0200	Data,	91	0.02992	FF55E3141B9442722EBB2E59776CE73834EA439CFFDFA9CA...
6	130.053632	0x0200	0x0100	Data,	211	0.009504	FF55E3141B9442722EBB2E59776CE73834EA439CFFDFA9CA...
7	130.084512	0x0090	0x0200	Data,	92	0.03088	FF55E5141B944277FEA72E76775795E5FFABCE725FD4AF39...
8	130.094016	0x0200	0x0100	Data,	212	0.009504	FF55E5141B944277FEA72E76775795E5FFABCE725FD4AF39...
9	130.126752	0x0090	0x0200	Data,	93	0.032736	FF55E7141B942E52FEA5795BFF579CBB3FA902C05FD4E739...
10	130.136256	0x0200	0x0100	Data,	213	0.009504	FF55E7141B942E52FEA5795BFF579CBB3FA902C05FD4E739...
11	130.16896	0x0090	0x0200	Data,	94	0.032704	FF55E9141B945C7CCD012A53A040B72B23A95A9C69FFA9D9...
12	130.180256	0x0200	0x0100	Data,	214	0.011296	FF55E9141B945C7CCD012A53A040B72B23A95A9C69FFA9D9...
13	130.209248	0x0090	0x0200	Data,	95	0.028992	FF55EB141B944282F7B9D973FD5CECBB3BB6539C5DDB39CE...
14	130.220224	0x0200	0x0100	Data,	215	0.010976	FF55EB141B944282F7B9D973FD5CECBB3BB6539C5DDB39CE...
15	130.251808	0x0090	0x0200	Data,	96	0.031584	FF55ED141B941087EFDB2E7677F2E738BDEECBB39EF739D9...
16	130.263104	0x0200	0x0100	Data,	216	0.011296	FF55ED141B941087EFDB2E7677F2E738BDEECBB39EF739D9...

We can see that this relay time is about 10ms is these few lines. Figure 61 plots the relay time of all relayed packets (6380 packets) and the mean value is 0.10959 with a standard deviation of 0.00323. Some relaying time appeared higher (at 40ms) because the initial packet from the audio board was not captured. In this case, the value represent the time difference from the last relayed packets. As the aggregation level was A2 (the payload is then 48 bytes), the audio packets are sent every 40ms by the audio source. Therefore, even if the original packet from the audio board was not captured, a time difference of about 40ms means that the relay jitter is very small.

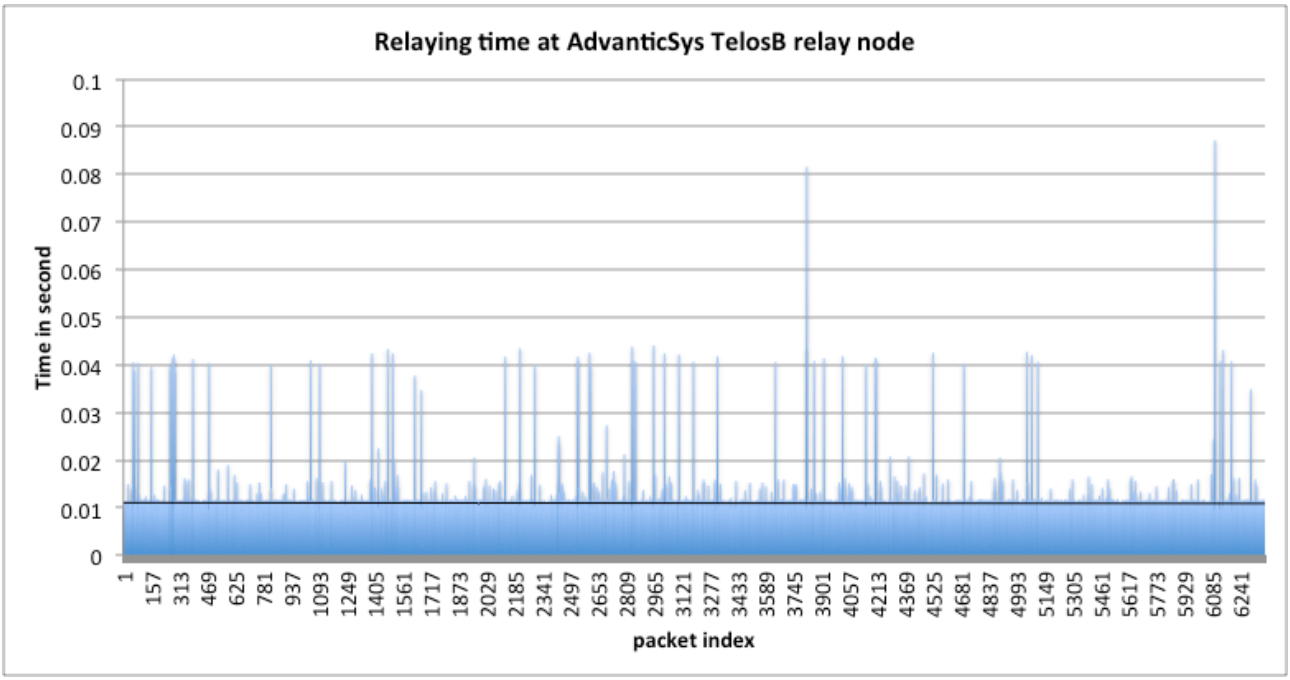


Figure 61: relay time of an AdvanticSys TelosB relay node, test#7

Now, compared to the relaying time shown previously in figure 4(bottom), the measures are quite consistent with what have been measured with the optimized version of our relay nodes.

Conclusion of benchmark tests in Geneva's HEPIA building

We summarize the main results of the benchmark tests performed in Geneva in the table below:

Geneva, HEPIA building			
test scenario	Pkt jitter at source	pkt jitter at relay	pkt loss rate
1-hop no occlusion	very small (1)	NA	0% - 8% (3)
1-hop occlusion	very small (1)	NA	16% - 81% (4)
2-hop	very small (1)	very small (5)(6)	5% - 30% (2)

1. The packet jitter at the source (AdvanticSys TelosB audio board), is very small and can be easily compensated at the destination with a very simple playout buffer.

2. At 2-hop or more, using relay nodes, non-LOS condition can be overcome, and reception quality can greatly be improved. This is particularly important in in-door environment as shown in the HEPIA building. However, the choice of the relay nodes can have a big impact of the performances.

3. In indoor environment, LOS transmissions (actually the distance between the source and the sink is quite small) show very low packet loss rate, similar to what can be found in open space environment.

4. In indoor environment, NLOS transmissions can rapidly become very difficult, decreasing dramatically the reception quality.

5. The packet relaying times measured with a promiscuous sniffer are consistent of what have been predicted in the previous deliverable. According to the maximum relaying capabilities, an appropriate aggregation level at the source can be used to reduce the packet losses at intermediate relay nodes.

6. The packet relay jitter was found again quite small and can be easily compensated at the destination with a very simple playout buffer.

Therefore, as a result of the benchmark tests, the Geneva's HEPIA test-bed is capable of supporting streamed audio in LOS transmission. in NLOS conditions, 1-hop transmission is not capable of providing a sufficiently small packet loss rate for an acceptable audio quality (packet loss rate much higher than 35%). Using 2-hop or more transmission can leverage the NLOS conditions and decrease the packet loss rate. However, the choice of the relay nodes is of critical importance to increase transmission quality, especially when transmitting from one floor to another. In all cases, the packet jitter at the source and at the relay nodes is very small.

6. Energy indicators

We also set-up some energy consumption measures in order to determine the cost of capturing and transmitting audio data on an intensive basis. We use facilities from the SIAME laboratory of University of Pau and 2 students performed the experimental measures on both the WaspMote audio mote and the AdvanticSys TelosB with the developped audio board.

Figure 62(left) shows the stabilized power supply used to power the sensor boards, and figure 68(right) shows the voltage measure station.

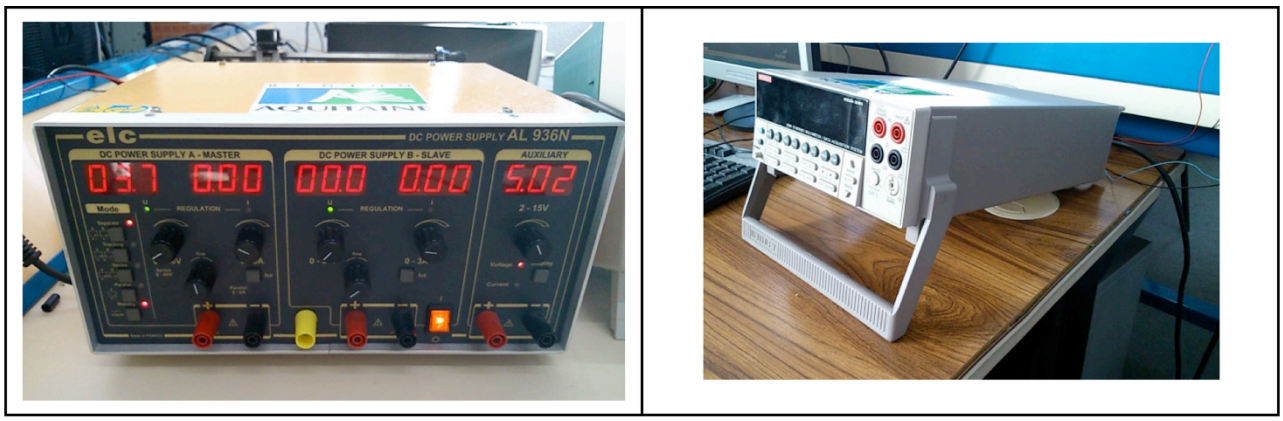


Figure 62: stabilized power supply (left), measure station (right)

Figure 63 shows for the WaspMote audio at 4kHz the cumulated energy consumption when the radio module is not plugged into the board. We can therefore measure the consumed energy when the board is idle and when the board is sampling at 4kHz. The behavior that is hard-coded into the mote is "idle" for 5s followed by "capture" for 15s.

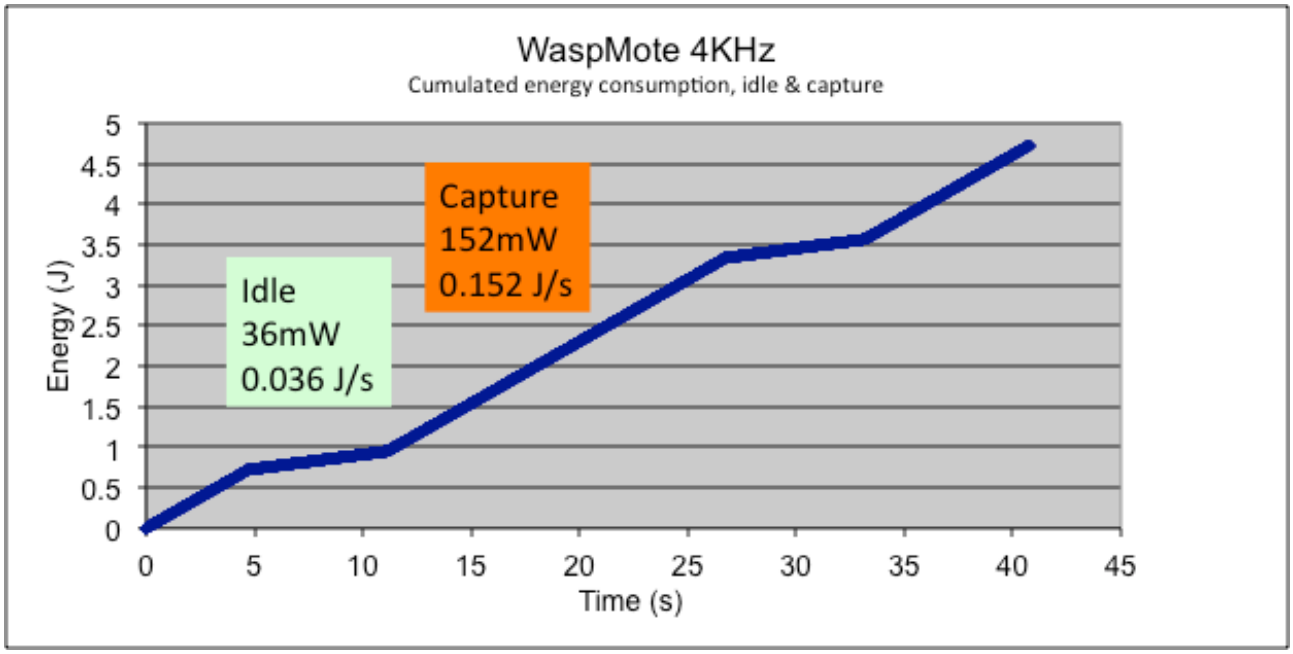


Figure 63: cumulated energy consumption, WaspMote audio, 4kHz, idle & capture

We then plugged the radio module (the XBee) and repeated the measures. Figure 64 shows the new cumulated energy consumption.

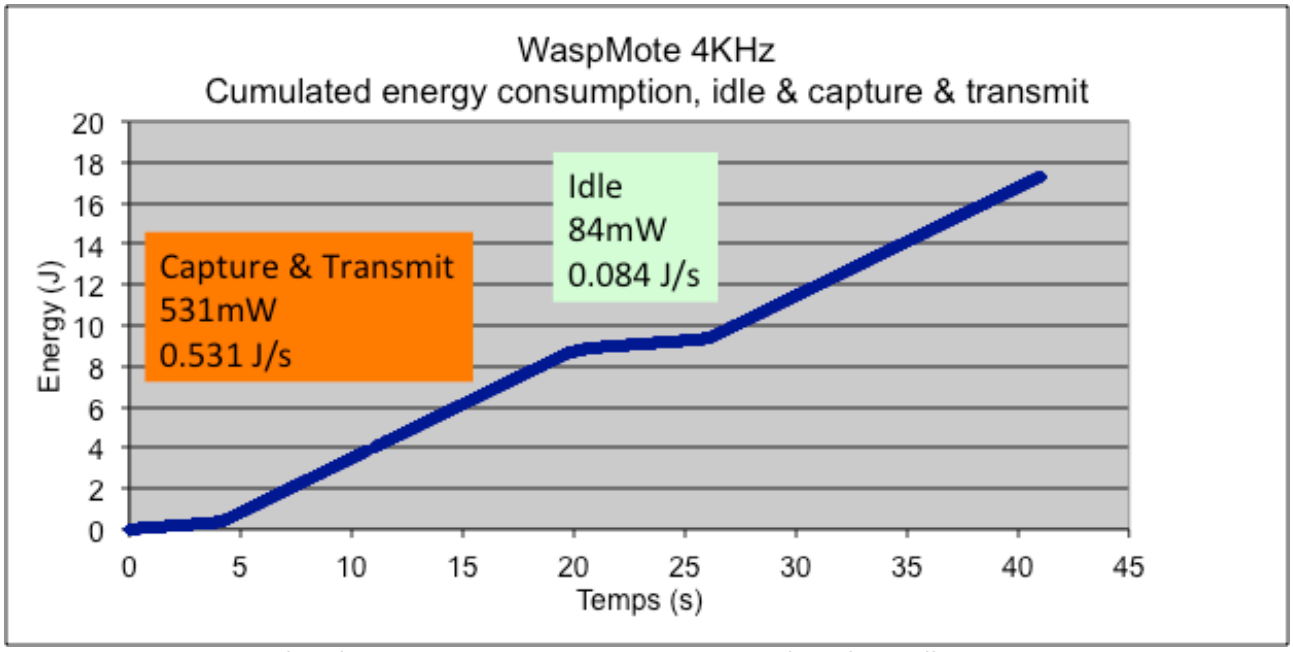


Figure 64: cumulated energy consumption, WaspMote audio, 4kHz, idle & capture & transmit

We can see that the energy consumed in "idle" mode when the radio is ON is higher than in the previous case, i.e. 0.084 J/s instead of 0.036 J/s. When capturing and transmitting, the WaspMote consumes about 0.531 J/s. With 2 AA batteries that usually are assumed to have an amount of energy of 18720 J, a simple prediction would allow for a continuous capture and transmission for about 9h and 47min.

We then use the 8kHz version. Figure 65 shows the cumulated energy consumption when the radio module is not plugged into the board.

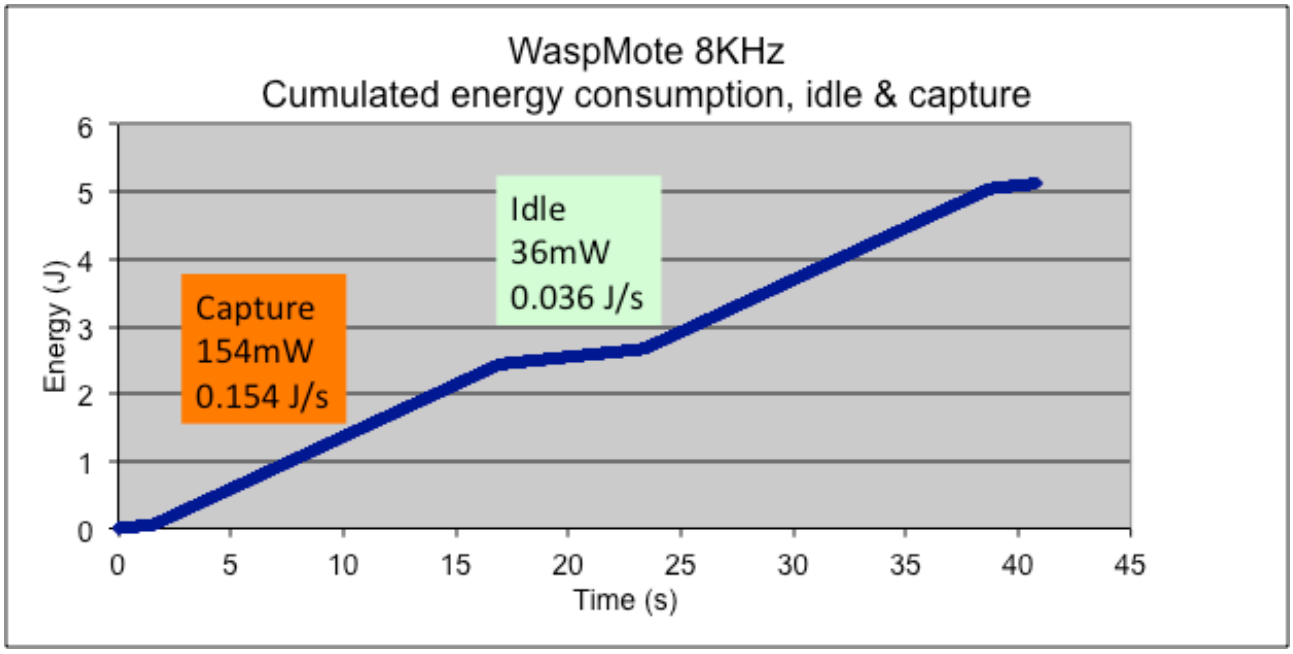


Figure 65: cumulated energy consumption, WaspMote audio, 8kHz, idle & capture

We can see that the consumed energy is very close to the 4kHz version meaning that sampling at 8kHz does not impact much on the board consumption. However, when the radio module is now plugged in, figure 66 shows the new cumulated energy consumption.

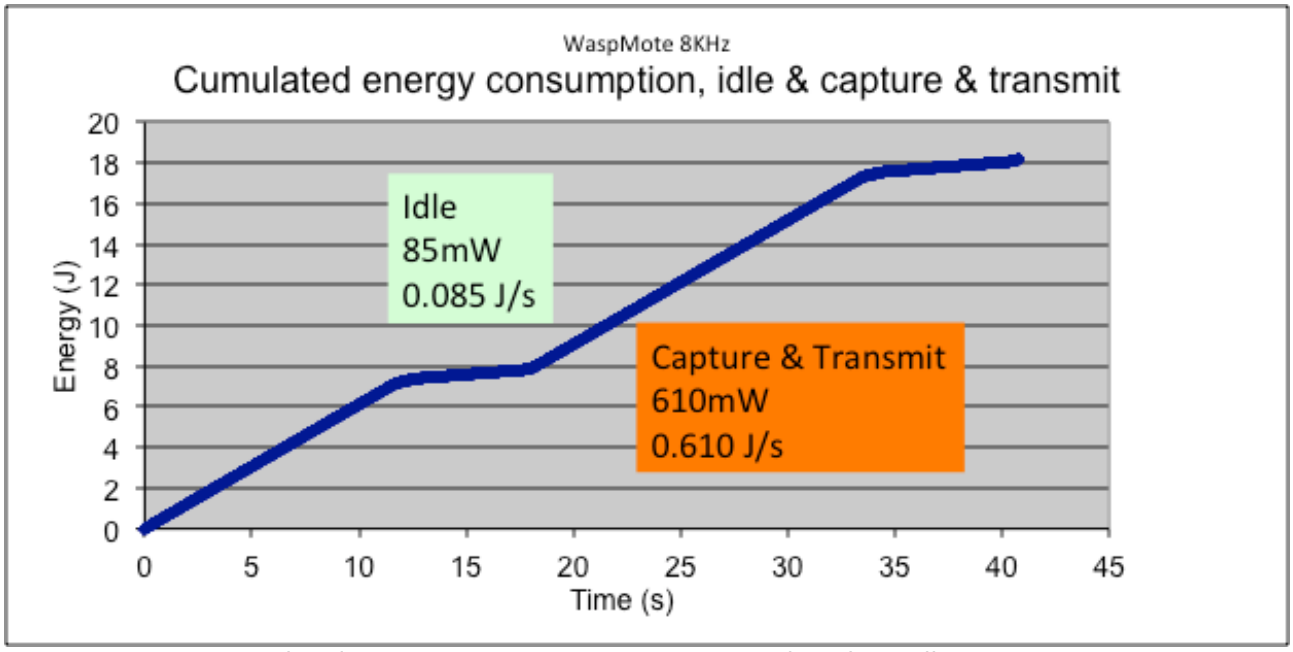


Figure 66: cumulated energy consumption, WaspMote audio, 8kHz, idle & capture & transmit

We can see that the "idle" consumption is the same than for the 4kHz version depicted in figure 64 and that the "capture & transmit" consumption raises to 0.610 J/s because of the larger amount of data transmitted. Again, a simple prediction with 2 AA batteries would give a continuous capture and transmission for about 8h and 30min.

In figure 67, we show the cumulated energy consumption for the AdvanticSys TelosB with the audio board. Since the radio module cannot be disconnected, we only have the case of "idle" and "capture & transmit".

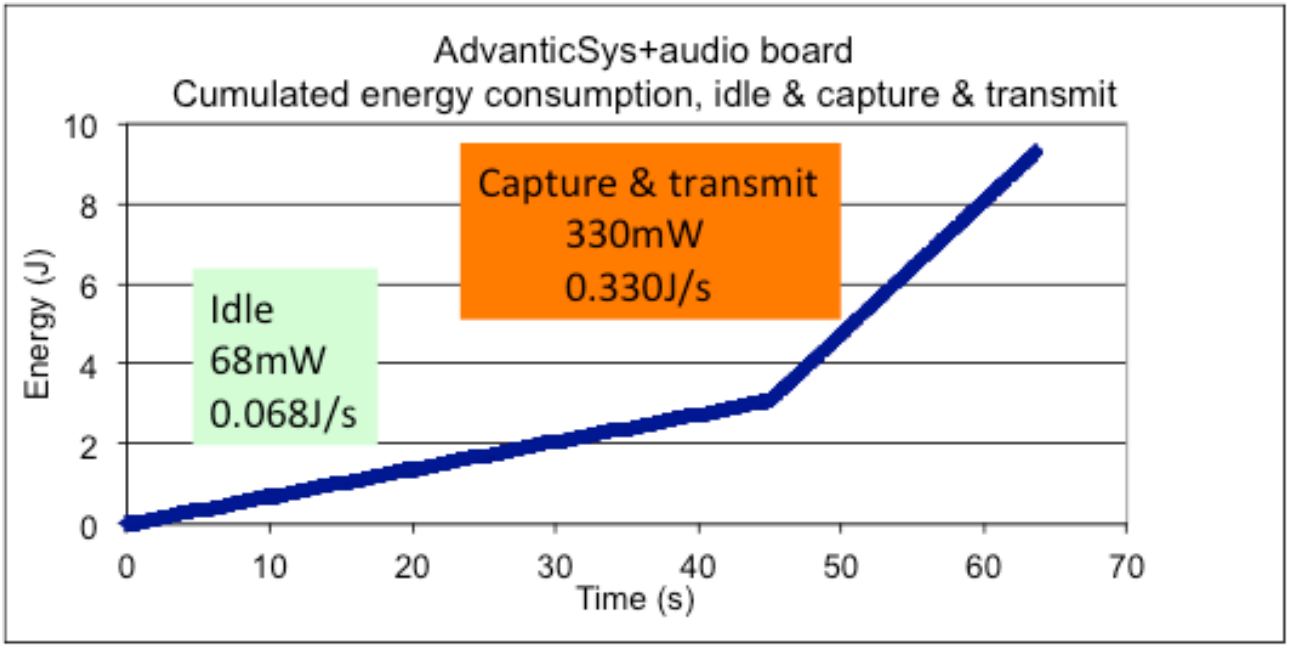


Figure 67: cumulated energy consumption AdvanticSys TelosB+audio board, idle & capture & transmit

Here, we can see that the "idle" consumption with the radio ON is lower than the consumption for the WaspMote in "idle" mode with the XBee on, i.e. 0.068 J/s instead of 0.085 J/s. The energy consumed while capturing and transmitting is also lower: 0.330 J/s with the audio board plugged in performing the real-time capture and speex compression. Again, a simple prediction would give a continuous capture and transmission for about 15h and 45min.

In figure 68, we plot the cumulated energy consumption for the WaspMote mote (those of Santander's SmartSantander test-bed) when relaying 100 packets of size 30 bytes. A traffic generator was used to generate 1 packet every 400ms.

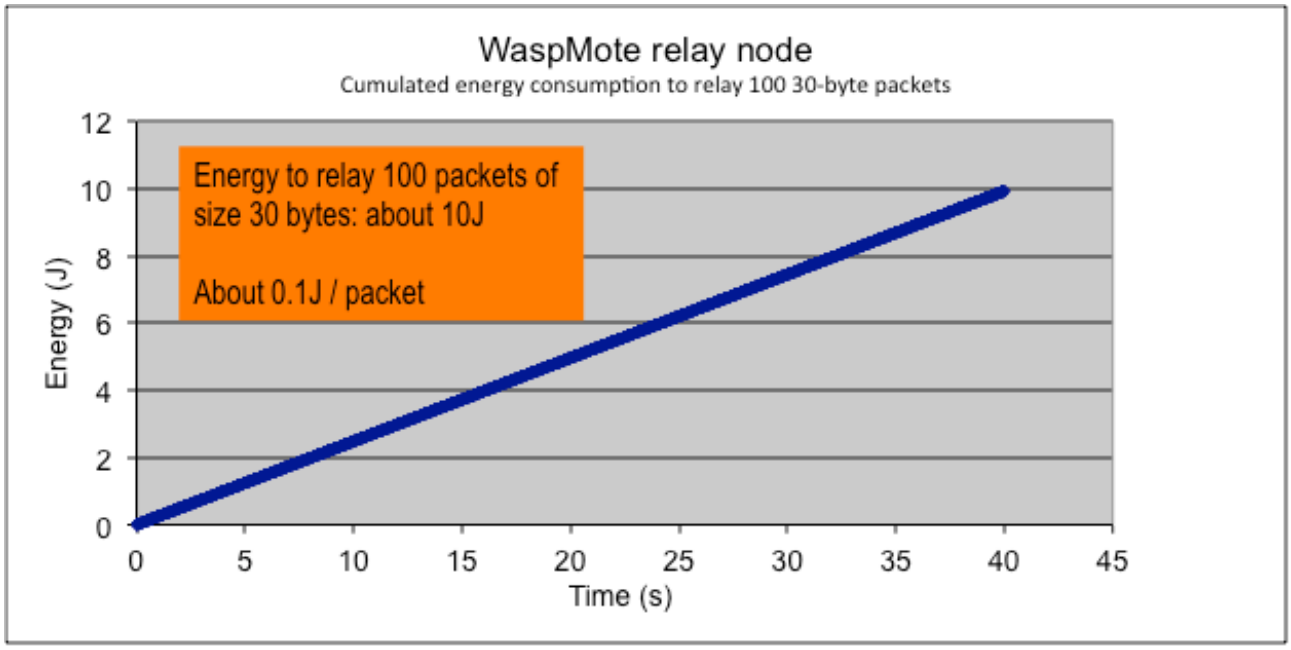


Figure 68: cumulated energy consumption WaspMote, relay

We found that the WaspMote needs about 0.1 J to relay a 30-byte packet. Using conservative assumption, the energy needed to relay 1 byte could be estimated at 0.0033 J. Table IV therefore can give an estimation of the relaying energy cost at various aggregation levels.

payload (bytes)	Aggregation level	relay consumption (J)
25	A1	0.0827
50	A2	0.1654
75	A3	0.2481
100	A4	0.3308

Figure IV: energy consumption WaspMote, relay

If we consider the 100-byte case and the A6 dedicated aggregation level (which capture 6 audio frames to send only 4 audio frames, giving a time window of 120ms), then the WaspMote relay node can relay for about 1h and 53min.

In figure 69, we plot the cumulated energy consumption for the AdvanticSys TelosB mote (those of Geneva's Hobnet test-bed) when relaying 100 packets of size 30 bytes. A traffic generator was used to generate 1 packet every 400ms.

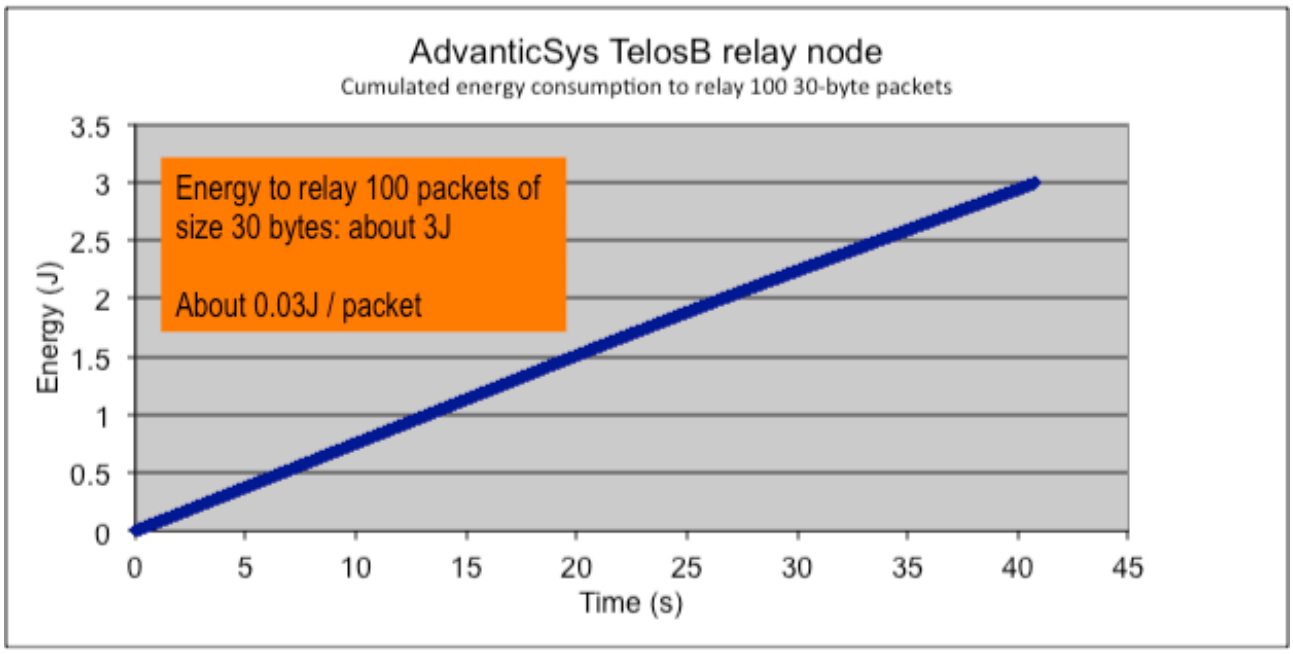


Figure 69: cumulated energy consumption AdvanticSys Telosb, relay

We found that the AdvanticSys TelosB needs about 0.03 J to relay a 30-byte packet. Using conservative assumption, the energy needed to relay 1 byte could be estimated at 0.001 J. Table V therefore can give an estimation of the relaying energy cost at various aggregation level.

payload (bytes)	Aggregation level	relay consumption (J)
25	A1	0.0250
50	A2	0.0500
75	A3	0.0750
100	A4	0.1000

Table V: energy consumption AdvanticSys, relay

If we consider the 100-byte case and the A4 aggregation level (which gives a time window of 80ms), then the AdvanticSys TelosB relay node can relay for about 4h and 10min.

7. Benchmarking other test-beds

Why doing a benchmark

The EAR-IT project working in various test beds in city environment (Santander) and in-door building (in Geneva), has demonstrated that promising applications can be developed using audio (traffic monitoring, security, energy efficiency, etc). Also using advanced audio codec (i.e. speex, codec2) we have demonstrated that even constrained network using 802.15.4 radio can be used for audio applications as audio streaming (the most constrained case) can be performed with only 2kbps bandwidth which is often available on these networks.

The project has now also defined the minimum condition for any test bed to be capable of hosting audio and audio related applications (see EAR-IT deliverable 1.2). The purpose of the benchmark procedure for other test-beds is to determine whether a given test bed is capable of providing the minimum requirements for supporting audio traffic.

Objectives of the benchmark

1. Determine whether a given test bed is capable of providing the minimum requirements for supporting audio traffic

2. Indicators and target values are given together with supporting documentation

What you need to do

1. Download the procedures and be ready to perform the tests on your test bed
2. Either use the developped audio mote or a simple traffic generator with a promiscuous packet sniffer that can also be downloaded
3. Determine if your test bed is "audio ready" by filled-in data in an excel sheet given where script can generate indicators which can be compared to minimum necessary
4. Audio source and audio hardware on TelosB can be borrowed to check on a real audio streaming conditions

Review of useful documents and EAR-IT deliverables

The proposed benchmark procedure is described in a set of slides "**WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds**", see ANNEX.C of this document. Read this document for detailed instructions on the benchmark procedure and the usage of the various tools that have been developped. The general benchmark methodology was also described in an earlier document "**WP1 Acoustic Test-bed Qualification/Qualify and Benchmark Test-beds for Acoustics in Deployment of Targeted Applications**". Our test-bed and various control software are also described in "**WP1 Acoustic Test-bed Qualification/Audio Test-bed Description**", see ANNEX.A of this document. Please refer to these documents as well as to deliverable "**WP1 Acoustic Test-bed Qualification/D1.2 : Miminium requirements for use of acoustic sensors**" that describe the developped audio board, the audio constraints and the purposes the test-bed benchmarking procedure. Additionally, there are a number of publications that you might find usefull as well:

1. C. Pham, P. Cousin, A. Carer, "Real-time On-Demand Multi-Hop Audio Streaming with Low-Resource Sensor Motes", Proceedings of IEEE SenseApp, in conjunction with LCN 2014, Edmonton, Canada, September 2014.
2. C. Pham and P. Cousin, "Benchmarking low-resource device test-beds for real-time acoustic data", Proceedings of the 9th International Conference on Testbeds and

Research Infrastructures for the Development of Networks & Communities (TridentCom'2014) , Guangzhou, China, May 5-7, 2014. Slides .pdf

3. C. Pham and P. Cousin, "Streaming the Sound of Smart Cities: Experimentations on the SmartSantander test-bed", Proceeding of the 2013 IEEE International Conference on Internet of Things (iThings2013), Beijing, China, August 20-23, 2013. Slides .pdf

Benchmarking procedure

The benchmarking procedure is explained in the EAR-IT web site:

http://www.ear-it.eu/audio-benchmarking

All resources such as scripting tools, Excel template files and communication tools are available for download.

ANNEX.B of this document reproduces the benchmarking procedure web pages.

ANNEX.C of this document shows the accompanying slides that explain further the benchmark procedure.

Call for Benchmark

A "Call for Benchmark" has also been issued to various scientific partners in order to validate the benchmark procedure and to have additional NETWORK indicators from other test-beds.

Preliminary results from Surrey test-bed

The University of Surrey accepted to conduct our benchmark procedure on their test-bed. The outcome was two-folds: first, the provided tools and benchmark procedure were validated by the Surrey team, second, the preliminary results consist of packet loss rate. Figure 70 below the packet interarrival time from the traffic generator. **The packet loss rate was found vey small, 5.07% (11 packet losses out of a total of 218), and fully compatible with speex audio requirements.**

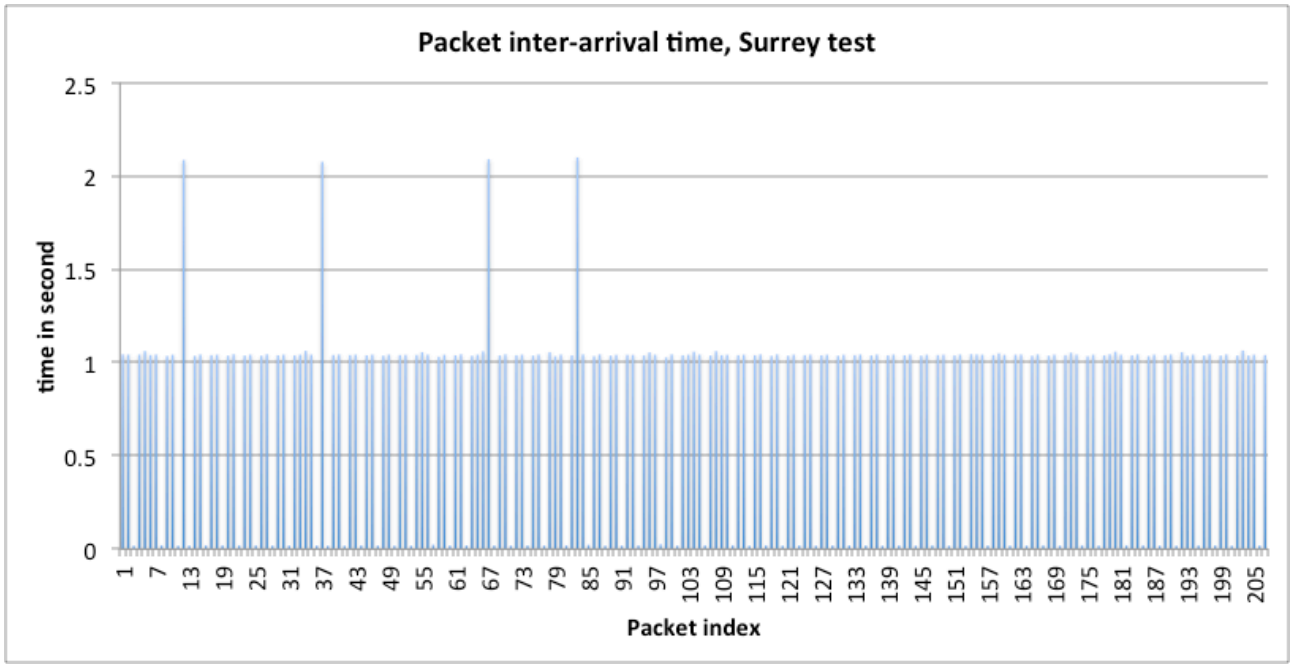


Figure 70 : packet inter-arrival time from the traffic generator, 100-byte packets

Figure 71 below shows the Surrey test settings.

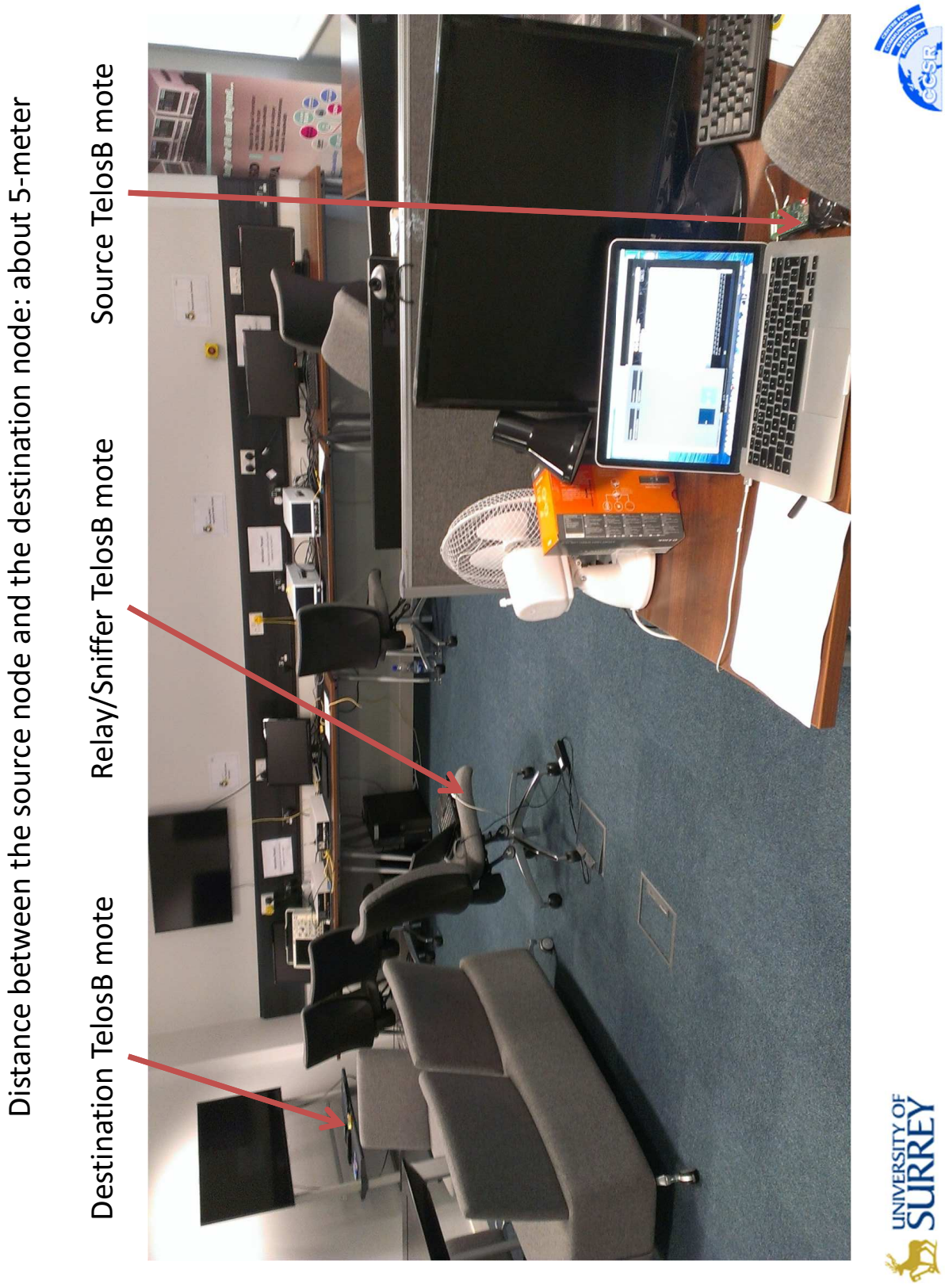


Figure 71 : test settings at University of Surrey

Preliminary results from EGM test-bed

EGM has a TST-based test-bed. TST motes are depicted in figure 72. They are designed and distributed by TST Sistemas in Spain.

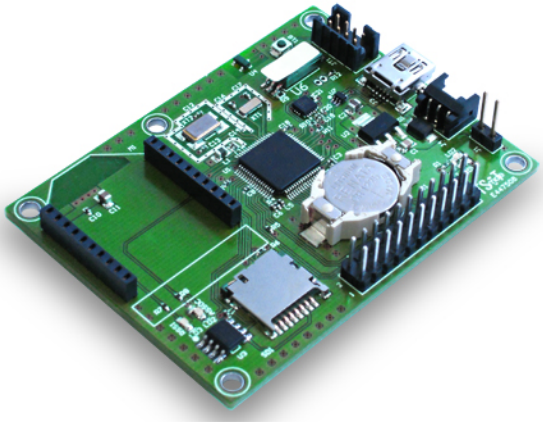


Figure 72 : TST mote

The TST mote main characteristics are summarized in Table VI below.

Microcontroller	32 bits STM with ARM Cortex-M3 core
Clock Frequency	72 MHz
Flash Memory	1 MB
RAM Memory	96 kB
Serial Interfaces	3 UART, 2 I2C, 1 SPI
Input / Output Ports	Up to 6 analog, up to 20 digital
Timer resolution	0.1 ms

Table VI: TST mote characteristics

The radio module is the XBee radio that was already studied. The main differences that we can expect from the TST motes are the much more powerful micro-controller clocked at a much faster clock rate than low-end sensor motes previously studied, i.e. Libelium WaspMote, Arduino and AdvanticSys TelosB.

The preliminary tests to verify the suitability of the EGM test-bed for acoustic data are performed on:

1. TST mote's sending capability, in case the audio board is connected to these motes as audio source,
2. TST mote's relaying capability in case these motes are deployed to relay acoustic data

Sending capability

Figure 73 shows the TST send overhead as the payload is increased. According to Table I**, the TST mote is perfectly capable of handling audio data from the audio board, i.e. one 24-byte packet every 20ms.**

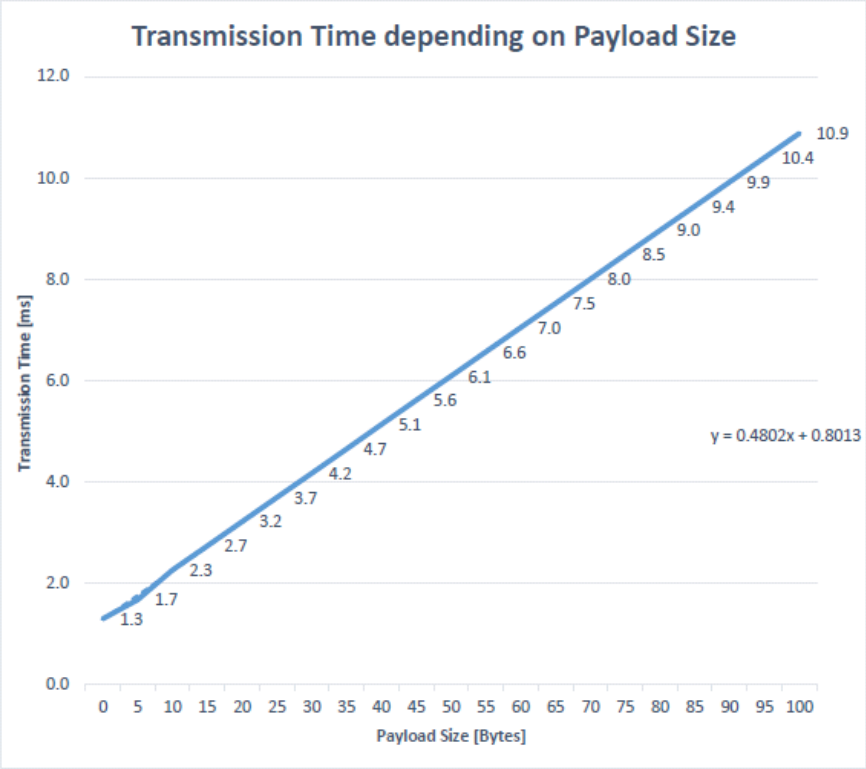


Figure 73 : TST mote sending overhead

Relaying capabiliy

Regarding the relaying capability for multi-hop audio streaming, figure 74 shows the relaying overhead as the payload is increased.

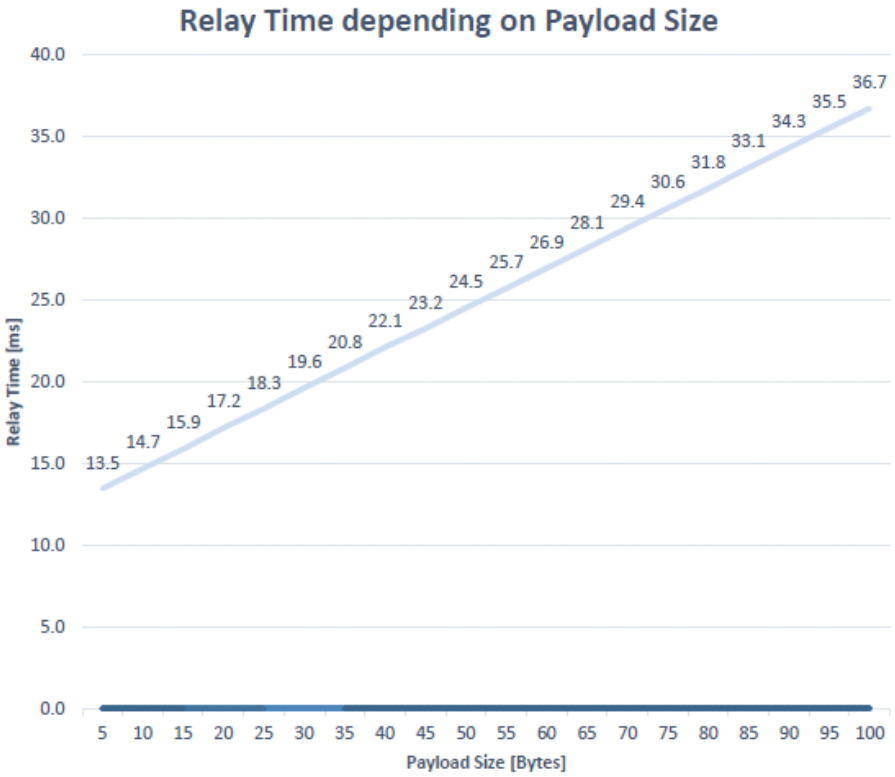


Figure 74 : TST mote relaying overhead

Once again, referring to Table I which indicates that a relay node must be able to relay a 24-byte packet in less than 20ms, we can see that the TST mote can satisfied this constraint. However, as it has previously been observed on the AdvanticSys TelosB motes, a mean relay time of 18.3ms for a 25-byte packet does not mean that all relays can be performed in less than 20ms. In this case, it is much safer to use A2 aggregation level to have a time window of 40ms. Figure 74 shows that the relay time for a 50-byte packet is on average 24.5ms, which is much lower than 40ms.

We can conclude that the EGM test-bed based on TST motes is fully capable of handling acoustic data.

8. Connecting the audio on other IoT platforms

The audio board has been designed to ease connectivity to other sensor mote platforms. The encoded audio stream is sent through an UART line (at 115200 or 38400 baud that could be configured on the audio board firmware). A 5V supply and a GND must be supplied, that are generally available on most sensor platforms. The host microcontroller should poll the corresponding serial input for data in order to get 20-byte audio frames every 20ms.

The audio board has been successfully connected to a Libelium WaspMote (the hardware that are deployed in Santander), see figure 72, and an Arduino MEGA 2560 board. ANNEX.D describes the procedure.

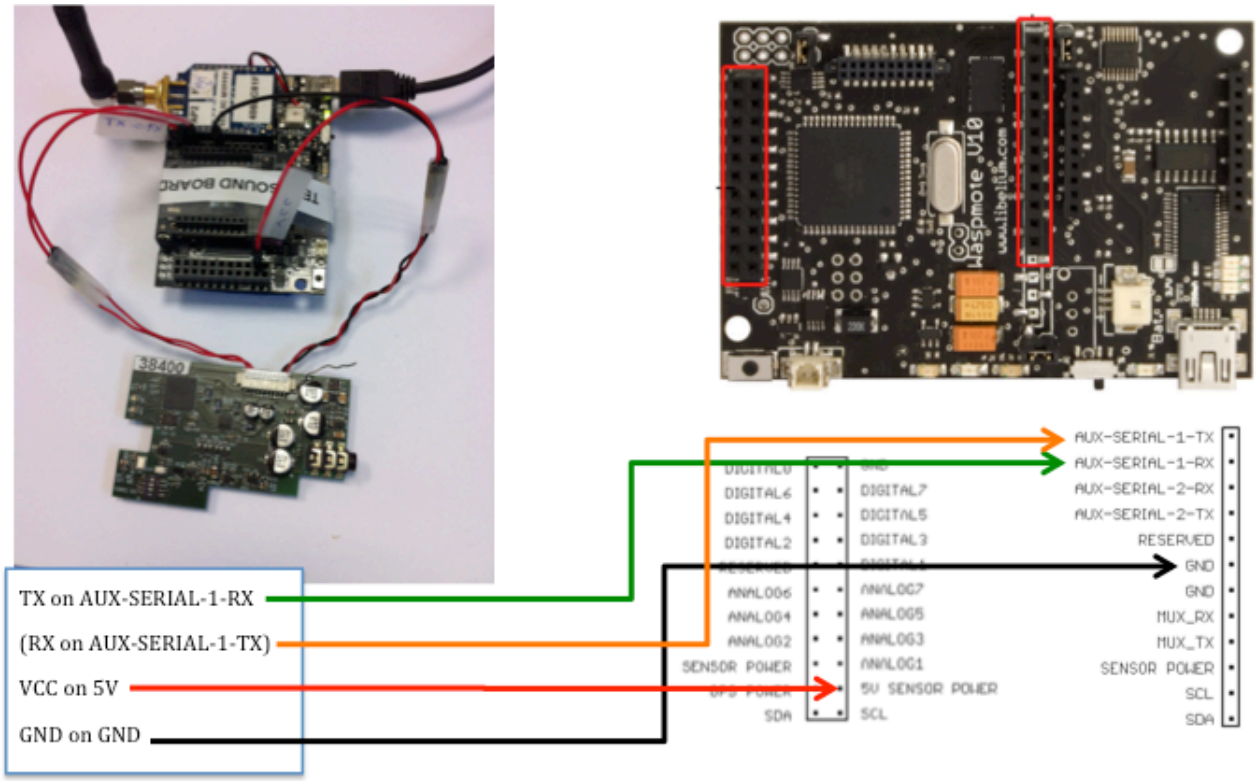


Figure 72 : connecting the audio board to a Libelium WaspMote

9. Summary and conclusions

Based on the results of deliverable 1.1 during the network qualification process, we developed an audio board with real-time sampling and encoding capabilities to allow for multi-hop audio streaming scenario. The audio board can be connected to most of sensor motes provided that a serial port is available.

We performed in-situ tests in both Santander and Geneva test-beds to determine in real conditions the network performances. For network indicators, we measured:

1. Packet jitter at the source
2. Packet loss rates at 1-hop
3. Packet loss rates at 2-hop
4. Packet relaying time at relay nodes
5. Packet relaying jitter at relay nodes

For energy indicators, we measured:

6. Energy consumption at the audio source
7. Energy consumption at the relay nodes

Summary of main results of the various tests

Network indicators

Santander, SmartSantander test-bed			
test scenario	Pkt jitter at source	pkt jitter at relay	pkt loss rate
1-hop LOS open space	very small (1)	NA	0% - 12% (2)
1-hop LOS urban	very small (1)	NA	35% (2)
1-hop NLOS urban	very small (1)	NA	60% - 92% (2)
2-hop open space	very small (1)	very small (6)(7)	5% - 23% (3)
2-hop urban	very small (1)	very small (6)(7)	53% (3)
Geneva, HEPIA building			
test scenario	Pkt jitter at source	pkt jitter at relay	pkt loss rate
1-hop no occlusion	very small (1)	NA	0% - 8% (4)
1-hop occlusion	very small (1)	NA	16% - 81% (5)
2-hop	very small (1)	very small (6)(7)	5% - 30% (3)

1. The packet jitter at the source, for both the WaspMote audio mote and the AdvanticSys TelosB audio board, is very small and can be easily compensated at the destination with a very simple playout buffer.

2. The packet loss rate at 1-hop in LOS condition, and when the distance of next hop is similar to what can be found in Santander, is very small. In non-LOS condition, for instance with buildings in-between, the packet loss rate can be very high: we for instance found packet loss rate as high as 92% with the developed audio board in a dense urban environment in non-LOS condition.

3. At 2-hop or more, using relay nodes, non-LOS condition can be overcome, and reception quality can greatly be improved. This is particularly important in in-door environment as shown in the HEPIA building. However, the choice of the relay nodes can have a big impact of the performances. In urban environment, the packet loss rate can still be high and more hops may be needed at the cost of higher latencies.

4. In indoor environment, LOS transmissions (actually the distance between the source and the sink is quite small) show very low packet loss rate, similar to what can be

found in open space environment.
5. In indoor environment, NLOS transmissions can rapidly become very difficult, decreasing dramatically the reception quality.

6. The packet relaying times measured with a promiscuous sniffer are consistent of what have been predicted in the previous deliverable. According to the maximum relaying capabilities, an appropriate aggregation level at the source can be used to reduce the packet losses at intermediate relay nodes.

7. The packet relay jitter was found again quite small and can be easily compensated at the destination with a very simple playout buffer.

Energy indicators

1. The energy consumption of the audio boards (both WaspMote and AdvanticSys TelosB) are found compatible with smart cities scenarios where nodes can be recharged at periodic moments in the day (at night for instance).
2. The relaying energy consumption was found to be the limiting factor in the system. However, in all cases, the relaying duration is larger than 1h. In emergency scenario where some minutes of streamed acoustic data are requested, we believe that 1h can be enough, especially if some advanced scheduling or audio source selection mechanism are implemented.

Conclusions

Santander test-bed

Ther SmartSantander test-bed in Santander is capable of supporting streamed audio both in open space and urban environment when LOS transmission is possible. in NLOS conditions, 1-hop transmission is not capable of providing a sufficiently small packet loss rate for an acceptable audio quality (packet loss rate much higher than 35%). Using 2-hop or more transmission can leverage the NLOS conditions and decrease the packet loss rate in urban environment. However, the choice of the relay nodes is of critical importance increase transmission quality and there are certainly many interesting issues to dynamically chose the right relay nodes. In all cases, the packet jitter at the source and at the relay nodes is very small.

Geneva test-bed

The Geneva's HEPIA test-bed is capable of supporting streamed audio in LOS transmission. in NLOS conditions, 1-hop transmission is not capable of providing a sufficiently small packet loss rate for an acceptable audio quality (packet loss rate much higher than 35%). Using 2-hop or more transmission can leverage the NLOS conditions and decrease the packet loss rate. However, the choice of the relay nodes is of critical importance to increase transmission quality, especially when transmitting from one floor to another. In all cases, the packet jitter at the source and at the relay nodes is very small.

10. References

[802154] IEEE Std 802.15.4™-2006.

[ADVAN] http://www.advanticsys.com/shop/wireless-sensor-networks-802154-mote-modules-c-7_3.html

[CC2420] ChipCon CC2420, 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. www.ti.com/lit/ds/symlink/cc2420.pdf

[DMDigi] XBee®/XBee-PRO® DigiMesh RF Modules product manual (90000991_E), Digi International Inc. January 6, 2012

[TELOSB] www.willow.co.uk/html/telosb_mote_platform.html and/or http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=252

[TINYOS] The TinyOS operating system. http://www.tinyos.net/

[XBeeDigi] XBee®/XBee-PRO® RF Modules product manual (90000982_G), Digi International Inc. August 1, 2012.

ANNEX.A: Review of software environment, tools and test hardware

Development environments

- Linux-based systems for higher flexibility and better interoperability
  - most of software tools are targeted for Unix
  - most of gateways devices are Linux-based (Meshlium, Beagle, Rasperry,…)
- When possible, avoid Java development and priviledge C, C++ and scripts (shell, python)

1

Standard IDE & software tools

- Libelium WaspMote
  - Libelium IDE (Arduino-based) & API development environment
- AdvanticSys TelosB
  - TinyOS 2.1.2 development environment
- Audio
  - Codec2 software (www.codec2.org): `c2enc`, `c2dec`
  - Speex software (www.speex.org): `speexenc`, `speexdec`
  - `sox` and `play` package (Linux)
- Serial & frame analysis
  - `minicom`, `cutecom`
  - `wireshark`

2

Customized speex audio tools

- Simple « pure » speex audio decoder without any header
  - Modified version of speex's `sampledec.c`
  - `speex_sampledec_wframing` : expects framing bytes
  - `speex_sampledec_nframing` : no framing bytes
- To get a « pure » speex audio encoded file without any header
  - Modified version of `speexdec.c` (yes `speexdec.c` and not `speexenc.c`) compatible with speex's `sampledec.c`

3

Development of dedicated tools

- Serial tools to read host computer serial port
  - `XBeeReceive` (C language)
  - `SerialToStdout` (python script)
    - 115200 baud version
    - 38400 baud version
- Communication tool to send control command packets
  - `XBeeSendCmd` (C language)
- Communication tool to send binary files
  - XBeeSendFile (C language)

4

XBeeReceive

- XBeeReceive
  - Main target is 802.15.4 XBee-based gateway
  - Translates XBee API frame
  - Reads from the serial port : `/dev/ttyUSB0, /dev/ttyS0, …`
  - Reconstructs file in binary mode (handles packet losses)
    - Assumes each packet with 4 bytes header: 2 bytes for file size & 2 bytes for offset
  - Can write to Unix stdout & can act as a transparent serial replacement
  - Can act in a data stream fashion: no header for packets

```
USAGE:    ./XBeeReceive -baud b -p dev -B -ap0 -v val –stdout –stream file_name
USAGE:    -baud, set baud rate, default is 38400
USAGE:    -p /dev/ttyUSB1
USAGE:    -B indicates binary mode. Assumes 4-bytes header for each pkt (that will be removed)
USAGE:    -framing expect for framing bytes 0xFF0x55 for binary data
USAGE:    -ap0, indicates an XBee in AP mode 0 (transparent mode) so do not decode frame structure
USAGE:    -v 77, use 0x77 to fill in missing value in binary mode
USAGE:    -stdout, write to stdout for pipe mode in binary mode
USAGE:    -stream, assumes no header & write to stdout for pipe mode in binary mode
USAGE:    file_name, name for saving binary file
```

5

SerialToStdout.py

- Simple python script to read serial port when no translation is needed
- Change baud rate and port as needed

```
import serial
import sys

ser = serial.Serial('/dev/ttyUSB0', 38400, timeout=0)

# flush everything that may have been received on the port to make sure
# that we start with a clean serial input
ser.flushInput()

while True:
    out = ''
    sys.stdout.write(ser.read(1024))
    sys.stdout.flush()
```

- `SerialToStdout.py` can be use instead of `XBeeReceive` with an XBee in transparent mode

6

XBeeSendCmd

- XBeeSendCmd
  - Main target is 802.15.4 XBee-based gateway
  - Send ASCII command with Xbee
  - Can be used to sent remote AT command to other Xbee module
  - Support DigiMesh firmware
  - Example
    - XBeeSendCmd -addr 0013a2004069165d "/@D0100#"

```
USAGE:     ./XBeeSendCmd -p dev [-L][-DM][-at] -tinyos -tinyos_amid id_hex -mac|-net|-addr|-b message
USAGE:     -p /dev/ttyUSB1
USAGE:     -mac 0013a2004069165d HELLO
USAGE:     -net 5678 HELLO
USAGE:     -addr 64_or_16_bit_addr HELLO
USAGE:     -b HELLO
USAGE:     -at to send remote AT command: -at -mac 0013a2004069165d ATMM
USAGE:     -L insert Libelium API header
USAGE:     -DM to specify DigiMesh firmware
USAGE:     -tinyos to forge a TinyOS ActiveMessage compatible packet (0x3F0x05 are inserted)
USAGE:     -tinyos_amid 6F, to set the ActiveMessage identifier to 0x6F (0x05 is the default)
```

7

XBeeSendFile

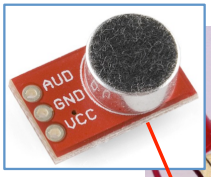
- XBeeSendFile
  - Main target is 802.15.4 XBee-based gateway
  - Send binary files with Xbee with controlled timing
  - Can use any packet size between 1 and 100 bytes
  - Can insert framing bytes, can introduce packet losses

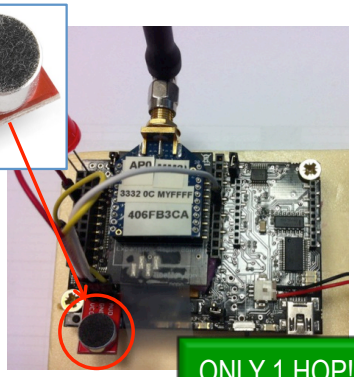
```
USAGE:     ./XBeeSendFile -baud baudrate -p dev -sensor -timing tpkt_us tserialbyte_us tafterradio_us -nw -fake
-drop rate -v val -fill -pktd -pktf -size s -stdout -mac|-net|addr|-b file
USAGE:     -baud 125000, 38400 by default
USAGE:     -sensor, will send image pkt to a sensor sniffer
USAGE:     -framing, will use framing bytes 0xFF0x55+SN for binary packets (e.g. audio)
USAGE:     -timing 50000 20 25000 by default
USAGE:     -nw, do not wait for TX status response
USAGE:     -fake, emulate sending. Will write in fakeSend.dat
USAGE:     -drop 50, will introduce 50 of packet drop. Useful with -fake
USAGE:     -v 77, use 0x77 to fill in missing bytes in lost packet
USAGE:     -fill, will fill missing bytes
USAGE:     -pktd, display generated XBee frames
USAGE:     -pktf, generate a pkt list file
USAGE:     -size 50, set packet size to 50 bytes
USAGE:     -stdout, write to stdout for pipe mode
USAGE:     -mac 0013a2004069165d
USAGE:     -net 5678
USAGE:     -addr 64_or_16_bit_addr, set either 64-bit or 16-bit dest. address
USAGE:     -b
```

8

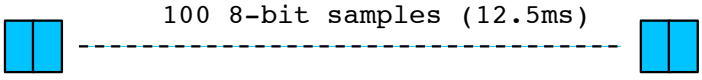
WaspMote+XBee in raw mode

- Electret mic with amplifier
- XBee in AP0 mode (transparent mode)
- 8-bit 4Khz sampling gives 32000bps
- 8Khz sampling gives 64000bps, requires custom API





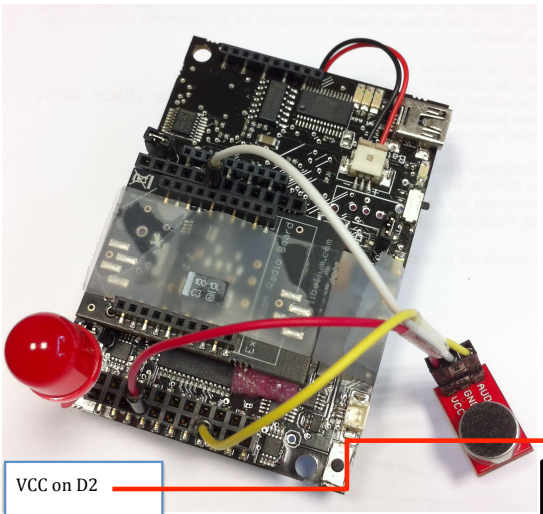
100 8-bit samples (12.5ms)

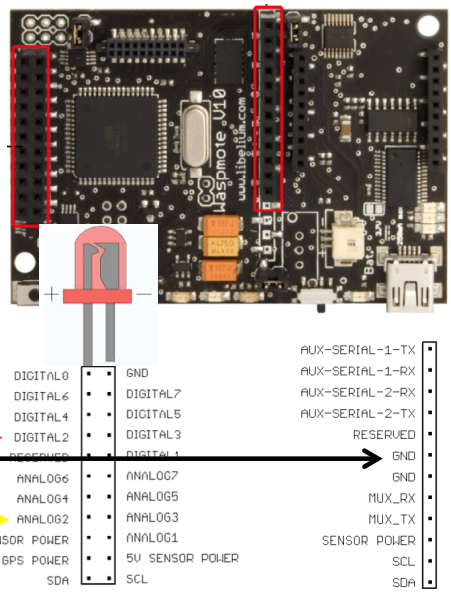


the sounds of smart environments

9

Details of pin connection

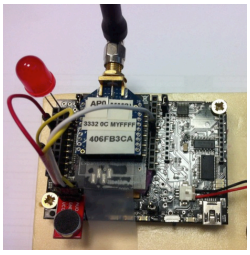




the sounds of smart environments

10

WaspMote test-bed: XBee gw AP0



```
void loop() {
    val = analogRead(ANALOG2) ; // read analog value
    val8bit = ((val >> 2) ) ; // convert into 8 bit

    // write on UART1, need an XBee module
    // with AP mode 0

    serialWrite(val8bit,1);
}
```

With XBee GW also in AP0 mode

```
4KHz sampling
> XBeeReceive -baud 38400 -ap0 -stdout dumb.dat | play --buffer 50 -t raw –r 4000 -u -1 –

8KHz sampling
> XBeeReceive -baud 125000 -ap0 -stdout dumb.dat | play --buffer 50 -t raw -r 8000 -u -1 -

Save raw data for off-line playing
> XBeeReceive -baud 38400 -ap0 -stdout dumb.dat > test.raw
> play -t raw –r 4000 -u -1 test.raw
```

Alternatively using SerialToStdout python script, at 38400 baud only

```
> python SerialToStdout | play --buffer 50 -t raw –r 4000 -u -1 –
```

11

XBee gateway in pkt mode (AP2)

- The receiving XBee module may need to be in packet mode (AP2) due to deployment constraints
- Adds overhead of XBee API frame decoding: 8KHz sampling may be not supported

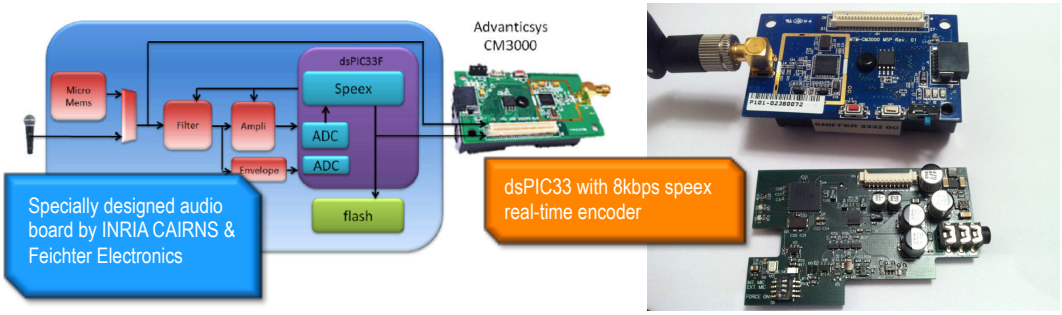
```
4KHz sampling
> XBeeReceive -baud 38400 –stream dumb.dat | play --buffer 50 -t raw –r 4000 -u -1 –


Save raw data for off-line playing
> XBeeReceive -baud 38400 –stream dumb.dat > test.raw
> play -t raw –r 4000 -u -1 test.raw
```

12

Multi-hop audio solution

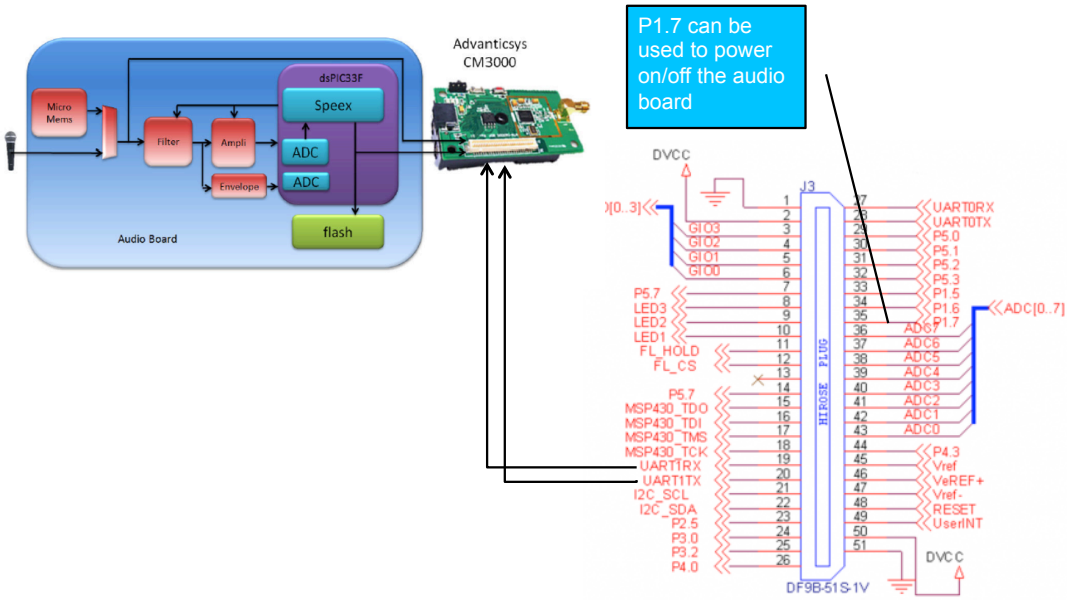
- Use dedicated audio board for sampling/storing/encoding at 8kbps



- Allows for multi-hop, encoded audio streaming scenarios

13

Details of pin connection



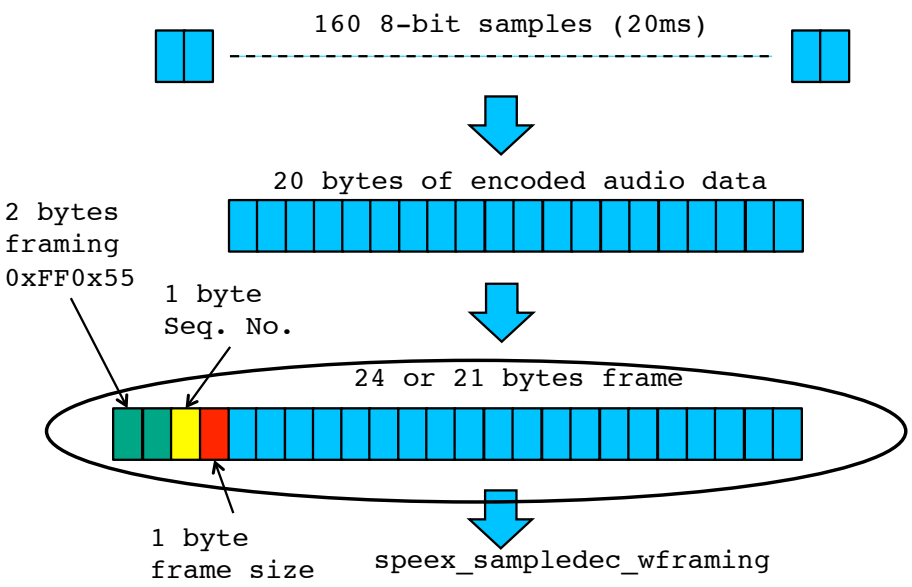
14

AdvanticSys+audio board

- The audio board captures 160 bytes (20ms) of raw audio and uses speex codec at 8kbps to produce 20 bytes to encoded audio data
- It sends the encoded audio data through an UART line to the host micro-controller
- The host micro-controller receives the encoded data and sends them wirelessly to the next hop
- The last hop is a base station that will forward the encoded audio into a speex audio decoder
- Output of the speex audio decoder is in raw format that can be feed into a player (`play`)

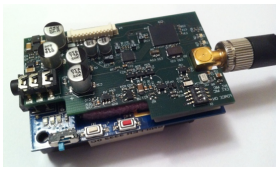
15

speex at 8kbps



16

AdvanticSys+audio board



```
async event void UartStream.receiveDone(uint8_t* buf,
     uint16_t len, error_t error){

     post sendMsg();
}
```

With AdvanticSys base station (115200 baud)

```
> python SerialToStdout | speex_sampledec_wframing | play --buffer 100 -t raw -r 8000 -s -2 -
```

With XBee GW in AP0 mode

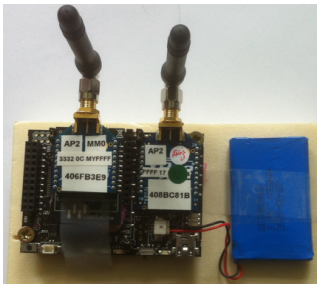

```
> XBeeReceive -baud 38400 -B -ap0 -stdout dumb.dat | speex_sampledec_nframing |
     play --buffer 100 -t raw -r 8000 -s -2 –
```

With XBee GW in AP2 mode (pkt mode)

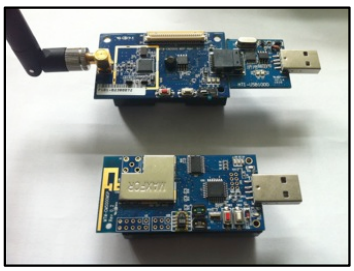
```
> XBeeReceive -baud 38400 -B -stream dumb.dat | speex_sampledec_nframing |
     play --buffer 100 -t raw -r 8000 -s -2 –
```

17

Relay nodes



LIBELIUM
WASPMOTE



ADVANTICSYS
CM5000, CM3000

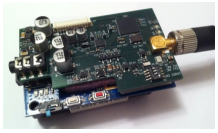
Fully configurable:

Destination node
Additional relay delay
Clock synchronization

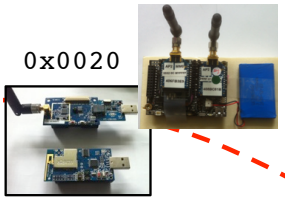
```
R0/1 enable/disable relay mode
D0013A2004086D828 set the 64-bit dest. mac addr
D0080 set the 16-bit dest. mac addr
```

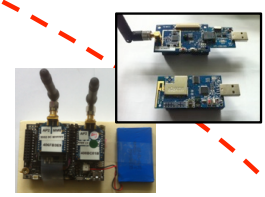
18

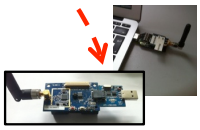
Multi-hop test-bed w/audio board



SPEEX AUDIO ENCODING
8KBPS







DECODE & PLAY
RECEIVED AUDIO

A1/2/3/4 aggregate audio frames
D0013A2004086D828 set the 64-bit dest. mac addr
D0080 set the 16-bit dest. mac addr
C0/1 power off/on the audio board

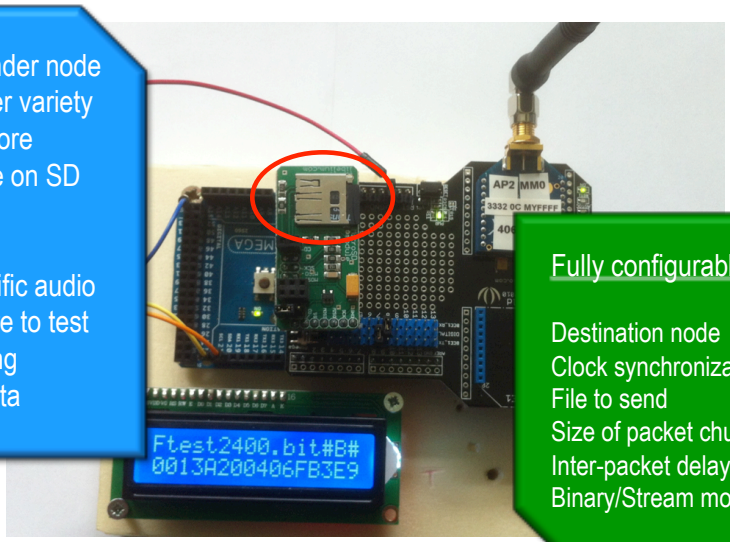
R0/1 enable/disable relay mode
D0013A2004086D828 set the 64-bit dest. mac addr
D0080 set the 16-bit dest. mac addr

19

Generic & controlled sender

Use a generic sender node to test with a larger variety of audio codec: store encoded audio file on SD card

Do not need specific audio encoding hardware to test quality of streaming encoded audio data

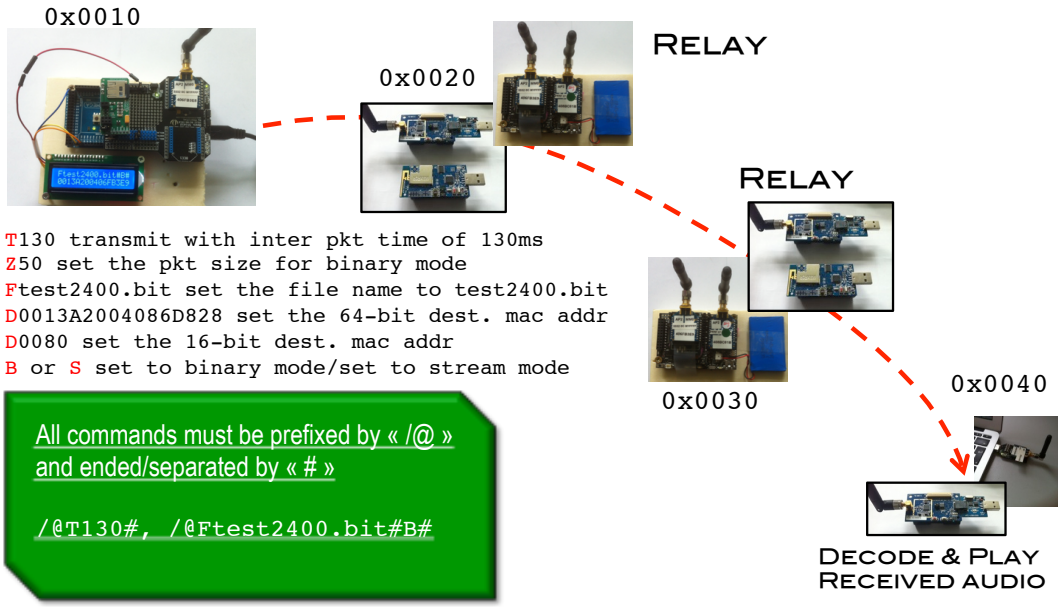


Fully configurable:

Destination node
Clock synchronization
File to send
Size of packet chunk
Inter-packet delay
Binary/Stream mode

20

Multi-hop test-bed w/generic sender



T130 transmit with inter pkt time of 130ms
Z50 set the pkt size for binary mode
Ftest2400.bit set the file name to test2400.bit
D0013A2004086D828 set the 64-bit dest. mac addr
D0080 set the 16-bit dest. mac addr
B or S set to binary mode/set to stream mode

All commands must be prefixed by « /@ »
and ended/separated by « # »

/@T130#, /@Ftest2400.bit#B#

21

the sounds of smart environments

codec2/speex with generic sender

- Use codec2/speex encoding software to produce encoded audio file
- Store encoded audio file (.bit/.spx) on SD card
- Configure the generic sender for sending the encoded audio file
  - Define packet size
  - Determine inter-packet send time
- Receive the encoded audio stream, decode the data and determine audio quality

22

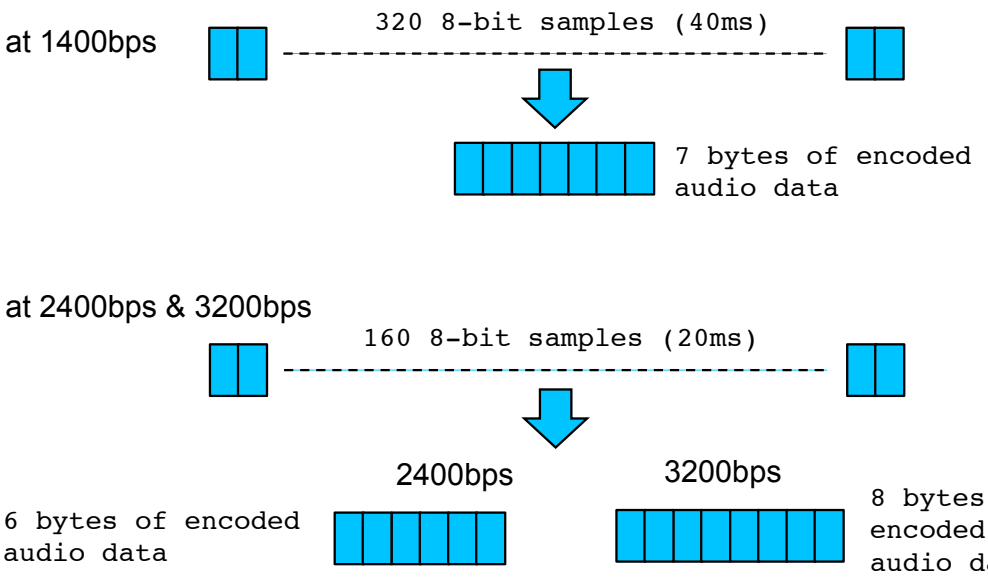
the sounds of smart environments

Produce encoded audio file: codec2

- Initial file: `test.raw` in 16-bit, signed
- Use `sox` to get 16-bit, signed if your raw file is not in this format
- Encode at 2400bps with
  - `c2enc 2400 test.raw test2400.bit`
- Store `test2400.bit` on SD card

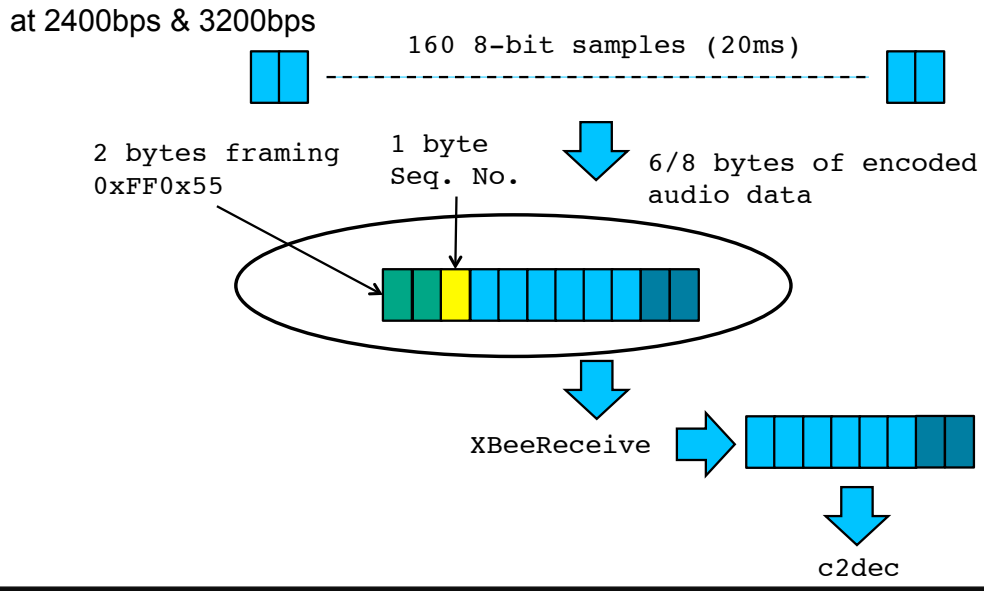
23

Codec2 encoding



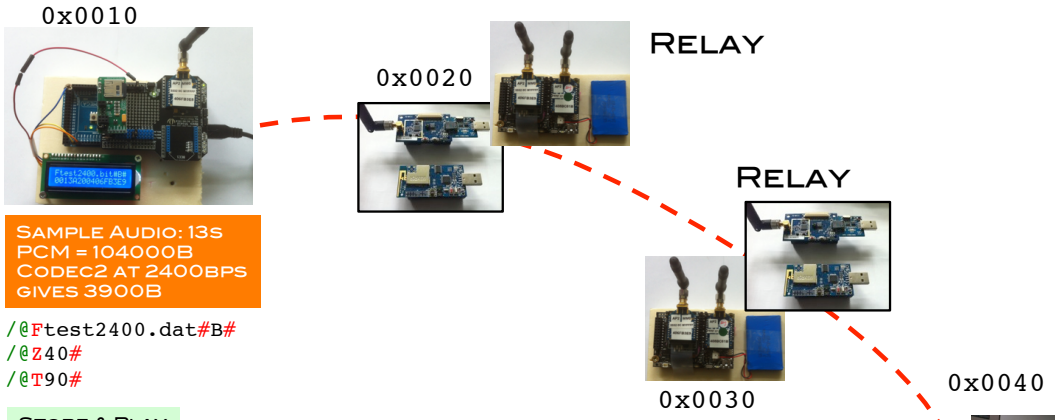
24

Codec2 at 2400bps & 3200



25

Multi-hop tests with codec2



SAMPLE AUDIO: 13S
PCM = 104000B
CODEC2 AT 2400BPS
GIVES 3900B

```
/@Ftest2400.dat#B#
/@Z40#
/@T90#
```

STORE & PLAY

```
> XBeeReceive -framing –B rcv-test2400.bit
> c2dec 2400 rcv-test2400.bit - | play -t raw -r 8000 -s -2 –
```

STREAMING

```
> XBeeReceive -framing –B -stdout rcv-test2400.bit | bfr -b1k -m2% - |
    c2dec 2400 - - | play -t raw -r 8000 -s -2 -
```

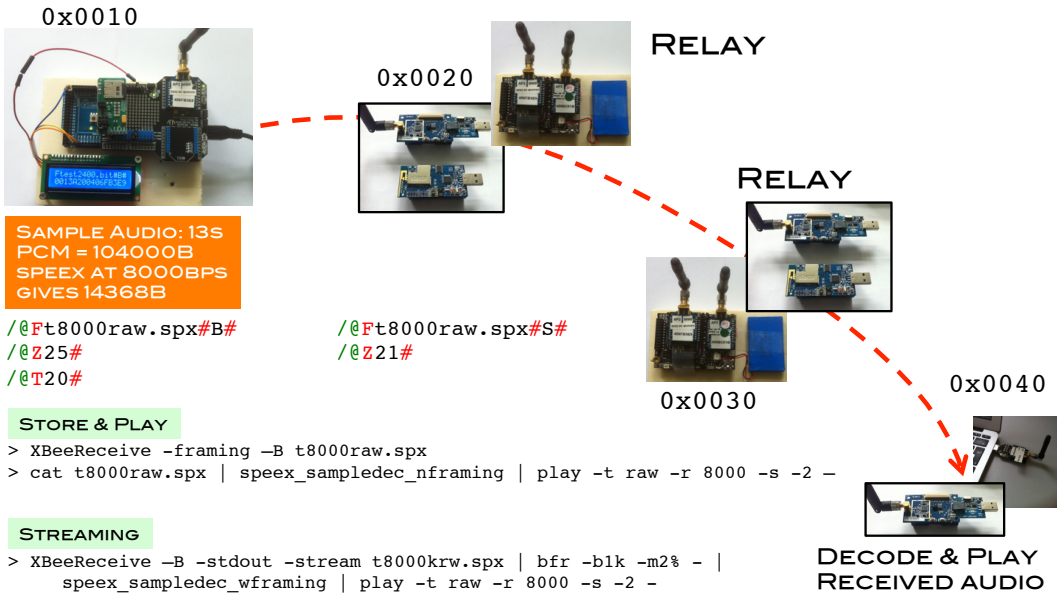
26

Produce encoded audio file: speex

- Initial file: `test.raw` in 8-bit unsigned or 16-bit signed
- Encode at 8000bps with
  - `speexenc --8bit --bitrate 8000 test.raw test8000.spx`
- Produce a raw speex byte stream with modified version of `speexdec`
  - `speexdec test8000.spx > t8000raw.spx`
- Store `t8000raw.spx` on SD card

27

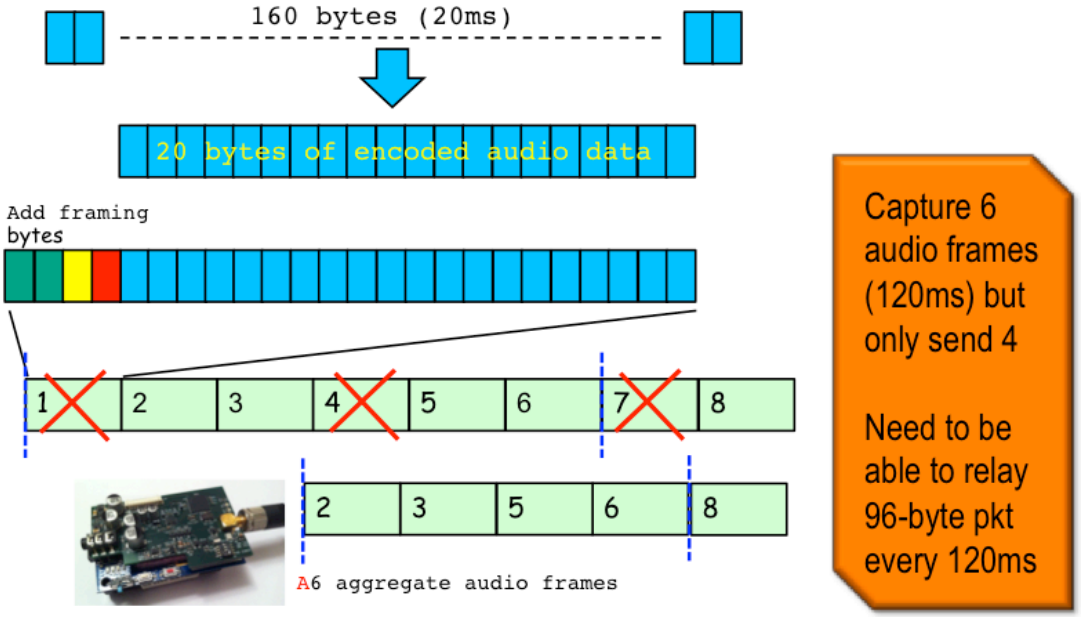
Multi-hop tests with speex



28

29

speex at 8kbps on slow relay nodes

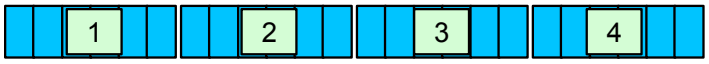


30

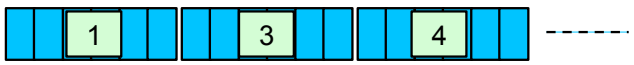
Apply packet loss rate

- Use XBeeSendFile to control
  - Timing between packet sending
  - Packet loss probability

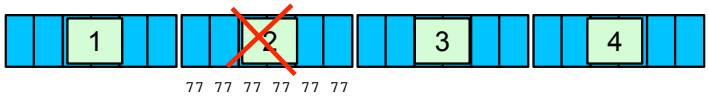
Codec2 2400bps, series of 6-byte encoded audio packets



> XBeeSendFile -fake -drop 25 -stdout test2400.bit > test2400-25loss.bit



> XBeeSendFile -fake -v 77 -fill -drop 25 -stdout test2400.bit > test2400-25loss-fill.bit



ANNEX.B: Benchmarking procedure for other test-beds

(reproduction of the benchmark procedure web pages)

Benchmarking IEEE 802.15.4 low-resource device test-beds for audio traffic: procedure & tools

as part of EAR-IT WP1: Acoustic Test-bed Qualification

C. Pham (LIUPPA laboratory, University of Pau, France & EGM) and P. Cousin (EGM, EAR-IT deputy project manager)

In the context of the FP7 EAR-IT project on acoustic surveillance in smart environments, this page describes and provides links to various tools for benchmarking low-resource device test-beds based on IEEE 802.15.4 connectivity.

last update: July 17th, 2014.

Why doing a benchmark ?

The EAR-IT project as working in various test beds in city (Santander) and with building (in Geneva) has demonstrated that nice applications can be developed using audio (traffic monitoring, security, energy efficiency, etc). Also using advanced audio codec (i.e. speex, codec2) we have demonstrated that even constrained network using 802.15 wireless network can be used for audio applications as audio streaming (the most constrained case) can be performed with only 2kbps bandwidth which is often available on these networks.

The project has now defined the minimum condition for any test bed to be capable of hosting audio and audio related applications (see EAR-IT deliverable 1.2). Doing this benchmark is easy and will allow your test bed to expand its usage for a broad range of amazing applications and research cases using acoustic.

Objectives of the benchmark

1. Determine whether a given test bed is capable of providing the minimum requirements for supporting audio traffic

2. Indicators and target values are given together with supporting documentation

What you need to do

1. Download the procedures and be ready to perform the tests on your test bed
2. Either use the developped audio mote or a simple traffic generator with a promiscuous packet sniffer that can also be downloaded
3. Determine if your test bed is “audio ready” by filled-in data in an excel sheet given where script can generate indicators which can be compared to minimum necessary
4. Audio source and audio hardware on TelosB can be borrowed to check on a real audio streaming conditions

Documents and EAR-IT deliverables

The proposed benchmark procedure is described in a set of slides "WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds". Read this document for detailed instructions on the benchmark procedure and the usage of the various tools that have been developped. The general benchmark methodology is also described in an earlier document "WP1 Acoustic Test-bed Qualification/Qualify and Benchmark Test-beds for Acoustics in Deployment of Targeted Applications". Our test-bed and various control software are also described in "WP1 Acoustic Test-bed Qualification/Audio Test-bed Description". Please refer to these document as well as to deliverable "WP1 Acoustic Test-bed Qualification/D1.2 : Miminium requirements for use of acoustic sensors" that describe the developped audio board, the audio constraints and the purposes the test-bed benchmarking procedure. Additionally, there are a number of publications that you might find usefull as well:

1. C. Pham, P. Cousin, A. Carer, "Real-time On-Demand Multi-Hop Audio Streaming with Low-Resource Sensor Motes", Proceedings of IEEE SenseApp, in conjunction with LCN 2014, Edmonton, Canada, September 2014.
2. C. Pham and P. Cousin, "Benchmarking low-resource device test-beds for real-time acoustic data", Proceedings of the 9th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom'2014) , Guangzhou, China, May 5-7, 2014. Slides .pdf
3. C. Pham and P. Cousin, "Streaming the Sound of Smart Cities: Experimentations on the SmartSantander test-bed", Proceeding of the 2013 IEEE International Conference on Internet of Things (iThings2013), Beijing, China, August 20-23, 2013. Slides .pdf

Benchmark tools

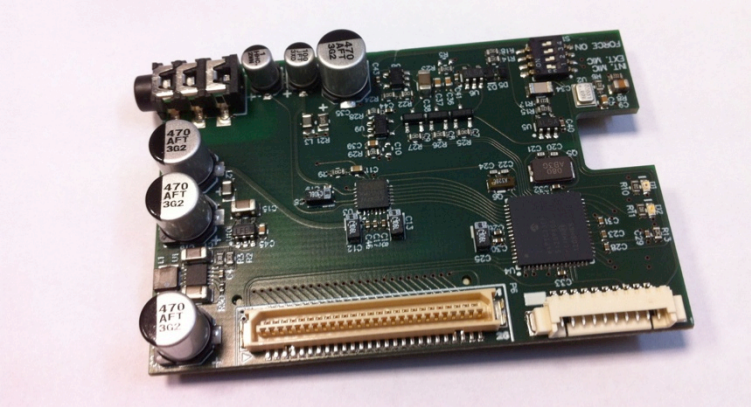
We developped in collaboration with INRIA CAIRNS and Feichter Electronics a daughter audio board with speex compression capability. The audio board is connected to a low-resource device, allowing real-time audio capture and compression. You actually don't need the audio board for the benchmark, but if you want to test real audio streaming on your test-bed, we can provide you with the audio mote, see the "Contact" section at the end of this page.

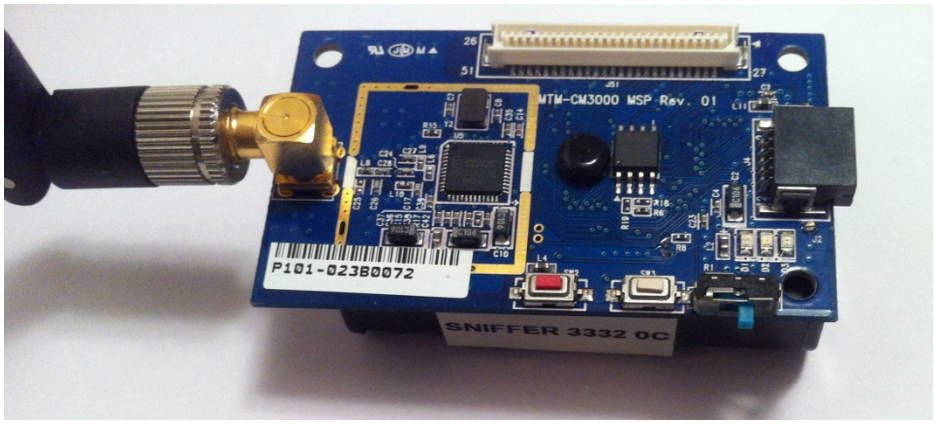
- **To use the full audio benchmark procedure with the developped audio board**, proceed with all steps from #1 to #6. You will be able to perform 1-hop audio streaming, determine relay capability of your test-bed for multihop audio and determine 1-hop packet loss rate in real audio conditions in your test-bed environment.

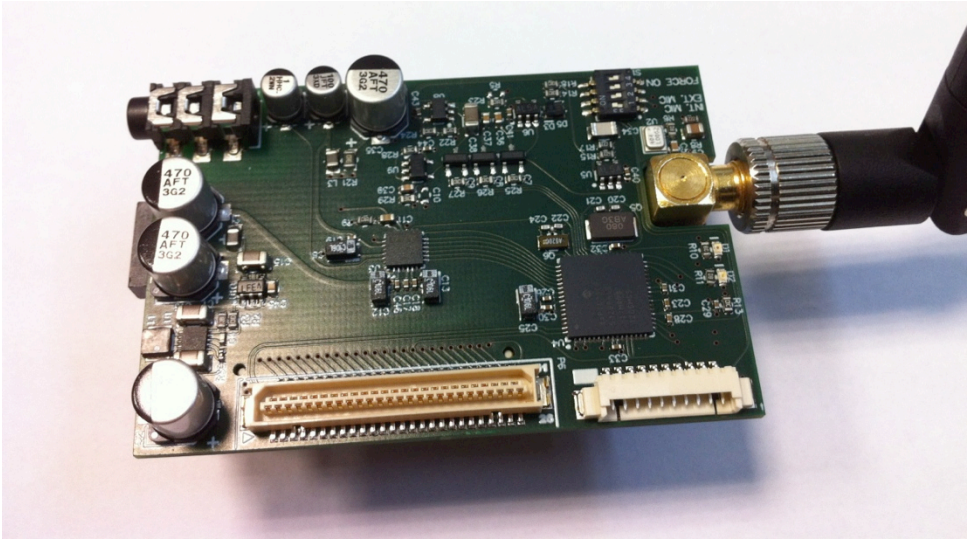
- **To perform a simple benchmark of your test-bed to determine the relaying capability** of your test-bed for multi-hop audio, you can proceed with steps #2, #5 and #6. Note that if you already have your own traffic generator and promiscuous packet analyser, you can only check in EAR-IT deliverable 1.2 whether your test-bed performances are adequate to support our audio mote traffic.

1/ Developped audio board and associated software

Figures below show the developped audio board that was initially designed to be connected to an AdvanticSys CM3000 mote (or CM3300 or CM4000), that will be referred to as the TelosB audio mote.







The control software for the Telosb audio mote gets the compressed data from the audio board and sends them to the next hop (a BaseStation or a relay node). The BaseStation is another TelosB mote that is connected to a Linux computer to act as a Sink. The BaseStation is not mandatory in the benchmark procedure but we can use it to listen in real-time to the audio stream. The archive for the source code of the TelosB audio and the BaseStation can be obtained here, all source code are under the TinyOS 2.1.2 operating system. Please refer to TinyOS installation instructions for setting up the TinyOS environment.

- archive for the source code of the TelosB audio and the BaseStation

The TelosB audio mote can be used to benchmark the test-bed for acoustic data. speex handles audio data in FRAME_SIZE. The value of FRAME_SIZE on the audio board is 160 bytes. Then encoding takes FRAME_SIZE bytes and compresses them is a number of encoded bytes. For 8000 bps rate, the encoded packet size is 20 bytes and the periodicity is 20ms. The BaseStation receives packets from the audio board. Each packet has frame delimiters (0xFF 0x55 SN) prior to the number of bytes (0x14=20) per packet in the output stream. SN will store an 8-bit sequence number. An example is shown below:

```
0xFF 0x55 0x00 0x14 .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
0xFF 0x55 0x01 0x14 .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
0xFF 0x55 0x02 0x14 .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..
```

To build the TelosB audio mote with IEEE 802.15.4 16-bit address of 0x0090:

```
> CFLAGS="-DNODE_SHORT_ADDRESS=0x0090 -DDEST_SHORT_ADDRESS=0x0100" make telosb
```

Then install with:

```
> make telosb reinstall bsl,/dev/ttyUSB0
```

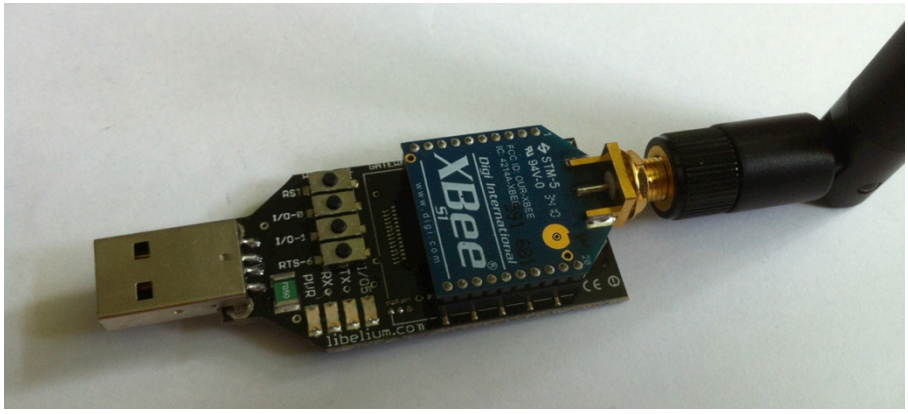
To build and install the BaseStation mote with IEEE 802.15.4 16-bit address of 0x0100:

```
> CFLAGS=-DNODE_SHORT_ADDRESS=0x0100 make telosb
> make telosb reinstall bsl,/dev/ttyUSB0
```

Then refer to "WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds" to see how the TelosB audio mote can be controlled wirelessly with the `XBeeSendCmd` tool.

2/ XBeeSendCmd

We use an XBee S1 from Digi (802.15.4, not ZigBee) as a radio gateway to send control messages. We provide the `XBeeSendCmd` tool that uses such gateway to send ASCII command. However, you can use any similar tools (or the X-CTU program provided by Digi) to send pure ASCII command sequence with an IEEE 802.15.4 radio module. Please refer to "WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds" for a list of ASCII control sequence used to control the TelosB mote.



The source code for `XBeeSendCmd` is available here:

- source code of the `XBeeSendCmd`

To compile:

```
> g++ -Wno-write-strings -o XBeeSendCmd XBeeSendCmd.c -lrt
```

Example (trigger audio capture and transmission at the TelosB audio mote):

```
> XBeeSendCmd -p /dev/ttyUSB0 -addr 0090 "@/C1#"
```

3/ Simple speex decoder at the sink

We also provide a simple speex decoder that waits for frame delimiters (0xFF 0x55) and that will decompress the audio stream into raw format to Linux's stdout. The source code for `speex_sampledec_wframing` is available here:

- source code of the `speex_sampledec_wframing`

To compile:

```
> gcc -DWITH_PKT_FRAMING -o speex_sampledec_wframing speex_sampledec.c -lspeex -lspeexdsp
```

See below for an example of usage

4/ 1-hop scenario with TelosB audio mote and BaseStation

Once you have the TelosB audio mote and the BaseSation mote, you can test by triggering the audio capture and listen in real-time to the audio stream. Here are the procedure:

1. Connect the XBee gateway to a computer (on /dev/ttyUSB0 for instance)
2. Connect the TelosB BaseStation to a computer (the same here, on /dev/ttyUSB1 for instance)
3. Run a python script that will continuously read the serial port for compressed audio data and that will forward these data to the speex decoder

```
> python 115200SerialToStdout.py /dev/ttyUSB1 | ./speex_sampledec_wframing
| play --buffer 100 -t raw -r 8000 -s -2 -
```

4. Power on the TelosB audio mote
5. Use `XBeeSendCmd`
    1. to activate the TelosB audio mote, you may need to send the control command twice

    ```
    > XBeeSendCmd -p /dev/ttyUSB0 -addr 0090 "@/C1#"
    ```

    2. to stop the TelosB audio mote

    ```
    > XBeeSendCmd -p /dev/ttyUSB0 -addr 0090 "@/C0#"
    ```

5/ Promiscuous packet sniffer for `wireshark`

We provide a promiscuous packet sniffer under TinyOS to be connected to the `wireshark` packet analyser. It's usage for the benchmark of test-bed is described in "WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds". The promiscuous packet sniffer is based on the TKN154 protocol stack and the `TestPromiscuous` or `packetsniffer` example. We improved `TestPromiscuous` to build a sniffer node. The source code is available here:

- archive for the source code of the promiscuous packet sniffer

To build and install the packet sniffer:

```
> CFLAGS="-DSNIFFER_CONF -DPCAP_SERIAL_OUTPUT" make telosb
> make telosb reinstall bsl,/dev/ttyUSB0
```

Then there is a simple python program that will continuously read the serial port and send data to `wireshark`. The mote will capture packets and will send pcap-formatted data to the serial port. More information on pcap format can be found here. The python program is TelosbToStdoutPcap.py

Then you can run the following command with your TelosB mote plugged in your computer on /dev/ttyUSB0:

```
> python TelosbToStdoutPcap.py | wireshark -k -i -
```

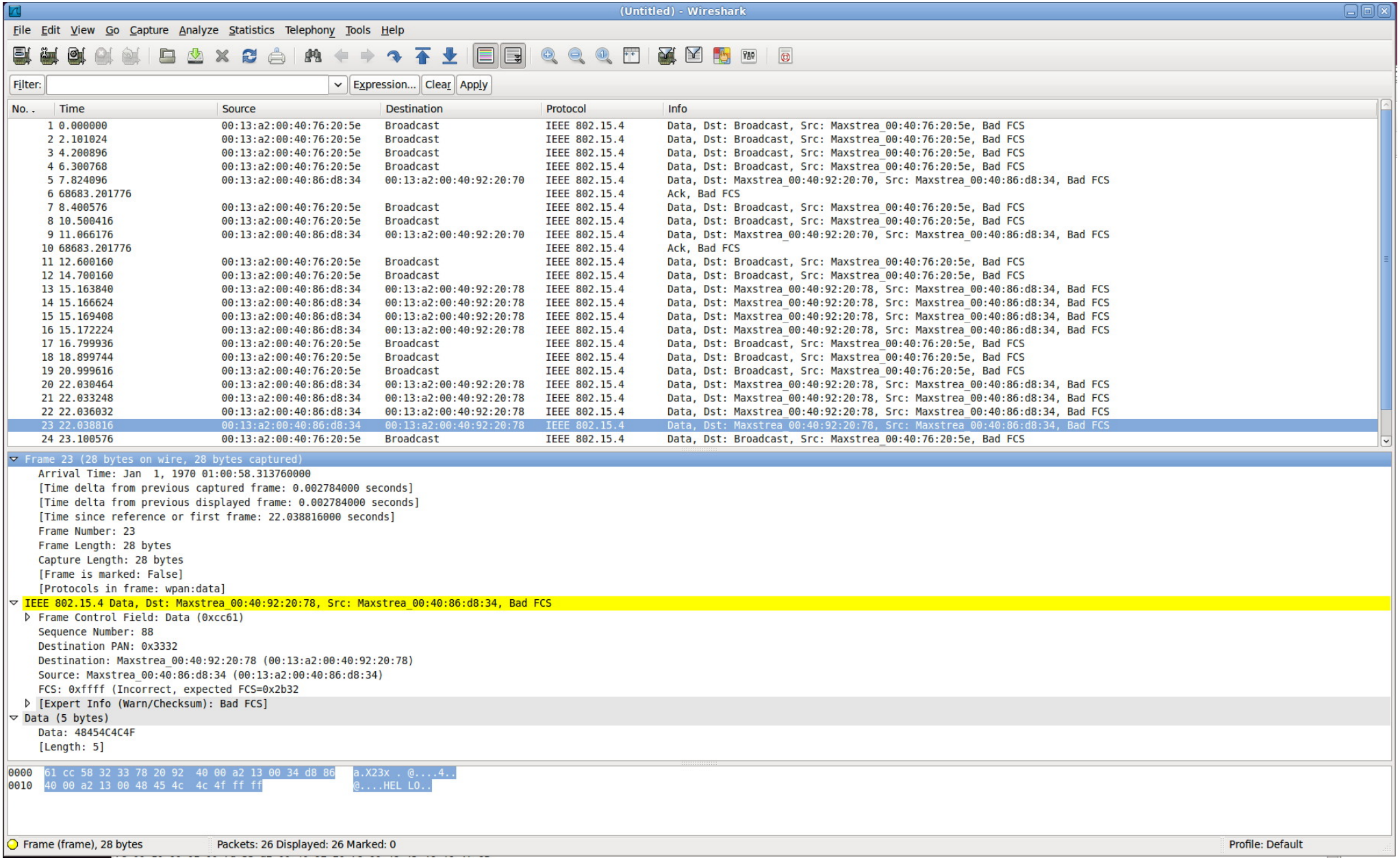
you may need to give sudo permission:

```
> python TelosbToStdoutPcap.py | sudo wireshark -k -i -
```

If running on /dev/ttyUSB1, just specifiy it in the command:

```
> python TelosbToStdoutPcap.py /dev/ttyUSB1 | wireshark -k -i -
```

You can see the graphical result below:


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: Expression... Clear Apply
No. . Time Source Destination Protocol Info
1 0.000000 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
2 2.101024 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
3 4.200896 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
4 6.300768 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
5 7.824096 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:70 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:70, Src: Maxstrea_00:40:86:d8:34, Bad FCS
6 68683.201776 IEEE 802.15.4 Ack, Bad FCS
7 8.400576 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
8 10.500416 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
9 11.066176 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:70 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:70, Src: Maxstrea_00:40:86:d8:34, Bad FCS
10 68683.201776 IEEE 802.15.4 Ack, Bad FCS
11 12.600160 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
12 14.700160 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
13 15.163840 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
14 15.166624 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
15 15.169408 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
16 15.172224 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
17 16.799936 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
18 18.899744 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
19 20.999616 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
20 22.030464 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
21 22.033248 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
22 22.036032 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
23 22.038816 00:13:a2:00:40:86:d8:34 00:13:a2:00:40:92:20:78 IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
24 23.100576 00:13:a2:00:40:76:20:5e Broadcast IEEE 802.15.4 Data, Dst: Broadcast, Src: Maxstrea_00:40:76:20:5e, Bad FCS
Frame 23 (28 bytes on wire, 28 bytes captured)
Arrival Time: Jan 1, 1970 01:00:58.313760000
[Time delta from previous captured frame: 0.002784000 seconds]
[Time delta from previous displayed frame: 0.002784000 seconds]
[Time since reference or first frame: 22.038816000 seconds]
Frame Number: 23
Frame Length: 28 bytes
Capture Length: 28 bytes
[Frame is marked: False]
[Protocols in frame: wpan:data]
IEEE 802.15.4 Data, Dst: Maxstrea_00:40:92:20:78, Src: Maxstrea_00:40:86:d8:34, Bad FCS
Frame Control Field: Data (0xcc61)
Sequence Number: 88
Destination PAN: 0x3332
Destination: Maxstrea_00:40:92:20:78 (00:13:a2:00:40:92:20:78)
Source: Maxstrea_00:40:86:d8:34 (00:13:a2:00:40:86:d8:34)
FCS: 0xffff (Incorrect, expected FCS=0x2b32
[Expert Info (Warn/Checksum): Bad FCS]
Data (5 bytes)
Data: 48454C4C4F
[Length: 5]
0000 61 cc 58 32 33 78 20 92 40 00 a2 13 00 34 d8 86 a.X23x . @....4..
0010 40 00 a2 13 00 48 45 4c 4c 4f ff ff @....HEL LO..
Frame (frame), 28 bytes
Packets: 26 Displayed: 26 Marked: 0
Profile: Default

You can see various scenarios in this snapshot:

1. broadcast packets do not need acknowledgment (see frame 1 for instance)
2. unicast packets need acknowledgment and the ACK is captured when the receiver is active (see frames 5 and 6 for instance)
3. unicast packet to an non-existing device will generate 1 transmission and 3 retransmissions (the default retransmission count in IEEE 802.15.4, see frame 13-16 for instance)

Limitations:

1. The timestamps for ACKs are normally incorrect from the SFD, but a turn around is proposed when using `wireshark`
    1. when a packet is an ACK packet, take the previous timestamp and add 192us (12 symbol=aTurnAroundTime)
    2. additionally adds 354us which is the ACK transmission time at 250kbps (11 bytes)
2. FCS is not valid so all frames will have bad checksum but it is not important as all captured frames already have good checksum for the radio module to accept them
3. When the sniffer is started while there are lot's of traffic, it may happen that the script sends a truncated frame read from the serial port resulting in an error such as "frame too long". You can have a more "secure" start by:
    1. press and **maintain** the reset button on the sniffer mote
    2. start the python script
    3. when `wireshark` is running, release the reset button. Since the radio module only accept valide frames (FCS is checked) starting the script well before the mote ensures that no truncated information from the serial port will be sent to `wireshark`.

Acknowledgments:

The original development tool for plugging a mote to `wireshark` has been provided by Pierre-Yves Lucas from University of Brest. He wrote a simple program to translate XBee API format to pcap format in order to be able to use `wireshark` with XBee module. We improved this idea by porting it to TelosB and MicaZ and CC2420 radio using TinyOS and TKN154 which is a much more powerful environment.

6/ Packet analysis script and Excel template

As described in "WP1 Acoustic Test-bed Qualification/Benchmarking procedure for other test-beds" here are:

1. An example of the `wireshark` capture converted into text format
2. The pkt-loss-rate awk script
3. The Excel template

Contact information

The TelosB audio board can be borrowed if you are willing to benchmark your test-bed. Please contact Philippe Cousin (EGM) from EAR-IT project.

ANNEX.C: Benchmarking procedure for other test-beds (slides)

Objectives of test-bed benchmarking

- Determine whether a given test-bed is capable of providing the minimum requirement for supporting audio traffic
  - Packet loss rate
  - Relaying capability
- Typical 1-hop packet loss rate need to be measured
- Performances of relay nodes need to be benchmarked for multi-hop audio
- EAR-IT support
  - Audio source nodes are provided
  - Source code of packet sniffer is available
  - Analysis script and Excel template are provided

the sounds of smart environments

1

Summary of audio characteristics at the source

Codec	Minimum sending rate
Raw 4KHz 8KHz	 100 bytes every 25ms 100 bytes every 12.5ms
Speex 8000bps A1 A2 A3 A4	 24 bytes every 20ms 48 bytes every 40ms 72 bytes every 60ms 96 bytes every 80ms

the sounds of smart environments 11

2

Procedure & tools for benchmarking a new test-bed

3

Benchmarking a new test-bed

- Determine 1-hop packet loss rate from audio source to either first relay node or gateway
  - Use maximum distance between audio source and first relay/gateway
- Determine performance of relay nodes
  - Packet relay latency
  - Packet relay jitter

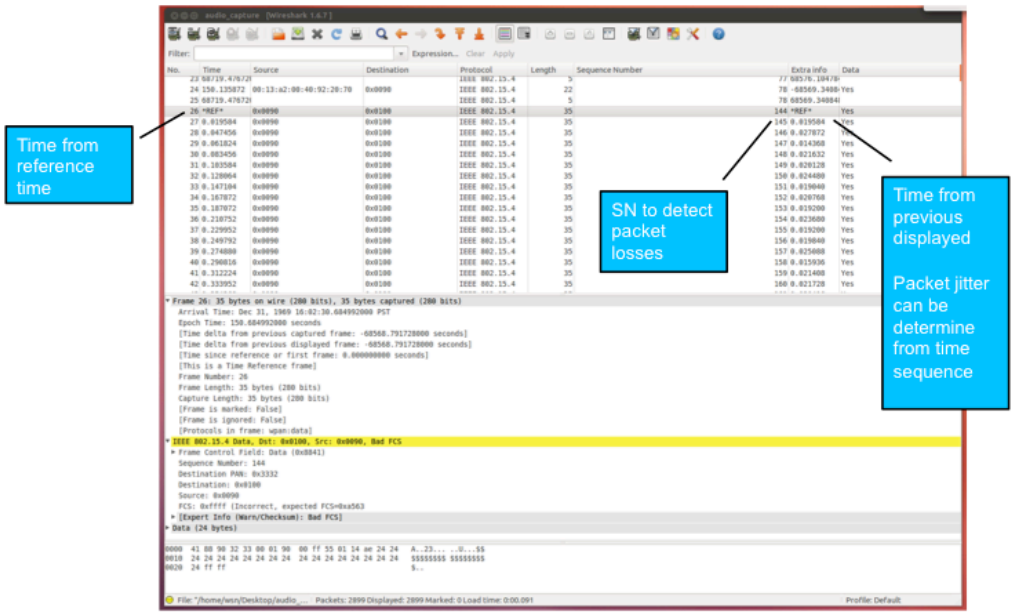
4

Frame analysis

- Use `wireshark` as frame analysis tool
- Use an AdvanticSys TelosB mote as promiscuous sniffer mote, connected to `wireshark` to display captured frames
- Frame sequence number and reception time can be visualized for statistic collection
  - Number of lost frames, frame loss rate
  - Frame transmission latencies
  - Frame jitter

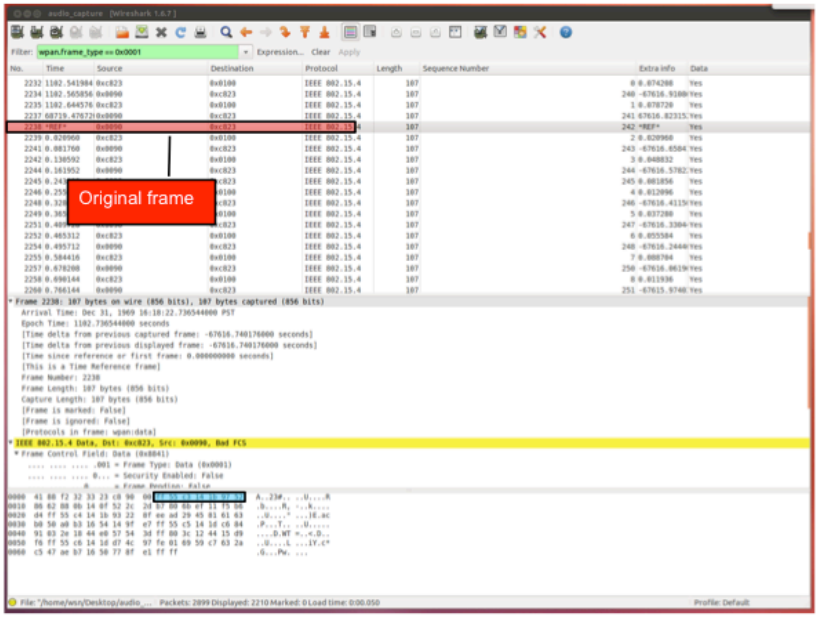
5

Example: packet losses & jitter



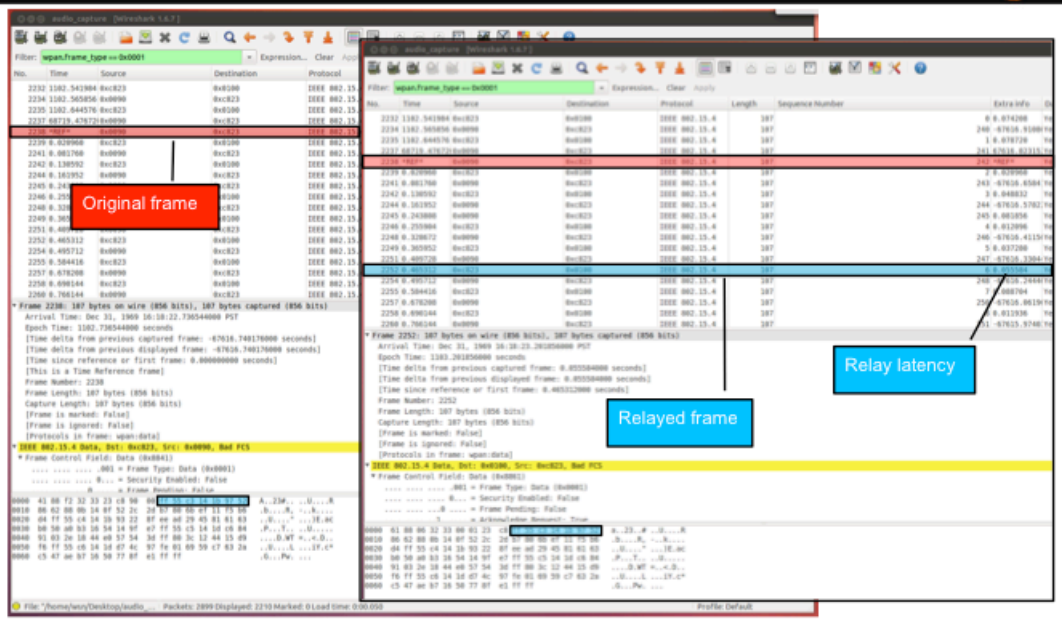
6

Example: relay latency



7

Example: relay latency



8

Illustration: 1-hop packet loss rate

Sniffer node will capture all frames in order to determine packet loss rate for typical/maximum 1-hop distance

9

Illustration: relay latency & jitter

Sniffer node will capture all frames (those from audio source and those from relay node) in order to measure relay latency & jitter

10

Simplified way to measure relay latency

- Instead of using the audio source to measure the relay latency, XBeeSendCmd can be used to send a number of packets of a given size at a given rate
- Example: broadcast 10 packets of 100 bytes, one every 500ms
  - `XBeeSendCmd -p /dev/ttyUSB0 -b -size 100 -n 10 -t 500`
- Use wireshark as previously described

11

Get statistics from wireshark captured frames

- Add custom columns info to have
  - IEEE 802.15.4 frame sequence number (wpan.seq_no)
  - Time from previously displayed frame
- Export the wireshark capture in text format, applying filters as needed (if filters, export only displayed frames)
- Also save the wireshark capture in pcap format for future usage as the pcap format stores all the information to apply additional filters if needed

12

Example: text file

No.	Time	Source	Destination	Protocol	Info	SN	Time
1	0.000000	0x0078	0x0000	IEEE 802.15.4	Data, Dst: 0x0000, Src: 0x0078, Bad FCS	1	0.000000
2	233.287936	00:13:a2:00:40:8b:c8:1b	0x0090	IEEE 802.15.4	Data, Dst: 0x0090, Src: Maxstrea_00:40:8b:c8:1b, Bad FCS	38	233.28793
3	233.288480			IEEE 802.15.4	Ack, Bad FCS	38	0.000544
4	233.945664	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	96	0.657184
5	234.071520	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	97	0.125856
6	234.195904	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	98	0.124384
7	234.321376	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	99	0.125472
8	234.445792	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	100	0.124416
9	234.570240	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	101	0.124448
10	234.694368	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	102	0.124128
11	234.820128	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	103	0.125760
12	234.944928	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	104	0.124800
13	235.069664	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	105	0.124736
14	235.194784	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	106	0.125120
15	235.318976	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	107	0.124192
16	235.442304	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	108	0.123328
17	235.568224	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	109	0.125920
18	235.693952	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	110	0.125728
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	111	0.122624
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	112	0.125344
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	113	0.124800
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	114	0.125024
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	115	0.125760
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	116	0.123040
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	117	0.124768
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	118	0.124960
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	119	0.125408
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	121	0.249952
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	122	0.123552
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	123	0.125632
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	124	0.124416
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	126	0.249088
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	127	0.126912
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	128	0.123168
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	129	0.124800
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	130	0.125984
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	131	0.123200
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	132	0.124800
			0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	133	0.125440
40	238.686784	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	134	0.124160
41	238.813440	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	135	0.126656
42	238.936928	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	136	0.123488
43	239.060896	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	137	0.123968
44	239.187200	0x0090	0x0100	IEEE 802.15.4	Data, Dst: 0x0100, Src: 0x0090, Bad FCS	138	0.126304

The first inter-arrival value is not correct, so replace by the value of the second frame (copy/paste)

Identify relevant part, removing lines associated to control messages (those used to start/stop the audio capture)

13

Simply determine packet loss rate

- Use the provided awk script to process the text file
- Be sure to have a text file with only the relevant frames (remove the control messages at the beginning and at the end of the captured trace)
- Example
  - `awk -f pkt-loss-rate.awk mytrace.txt`

14

Awk results

```
resources — bash — 120×24
MacBookProRetina-de-Congduc-Pham:resources cpham$ awk -f pkt-loss-rate.awk 03-392-meshlium-audio-board-A6.txt
--------------------
4 233.945664 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 96
nb packet: 1 lost: 0 total lost: 0
5 234.071520 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 97
nb packet: 2 lost: 0 total lost: 0
6 234.195904 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 98
nb packet: 3 lost: 0 total lost: 0
7 234.321376 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 99
nb packet: 4 lost: 0 total lost: 0
8 234.445792 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 100
nb packet: 5 lost: 0 total lost: 0
9 234.570240 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 101
nb packet: 6 lost: 0 total lost: 0
10 234.694368 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 102
nb packet: 7 lost: 0 total lost: 0
11 234.820128 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 103
nb packet: 8 lost: 0 total lost:
12 234.944928 0x0090 0x0100 IEEE
nb packet: 9 lost: 0 total lost:
13 235.069664 0x0090 0x0100 IEEE
nb packet: 10 lost: 0 total lost
14 235.194784 0x0090 0x0100 IEEE
nb packet: 11 lost: 0 total lost
```

Processes each line and shows the packet number, the # of lost packets at that stage and the total # of lost packets so far

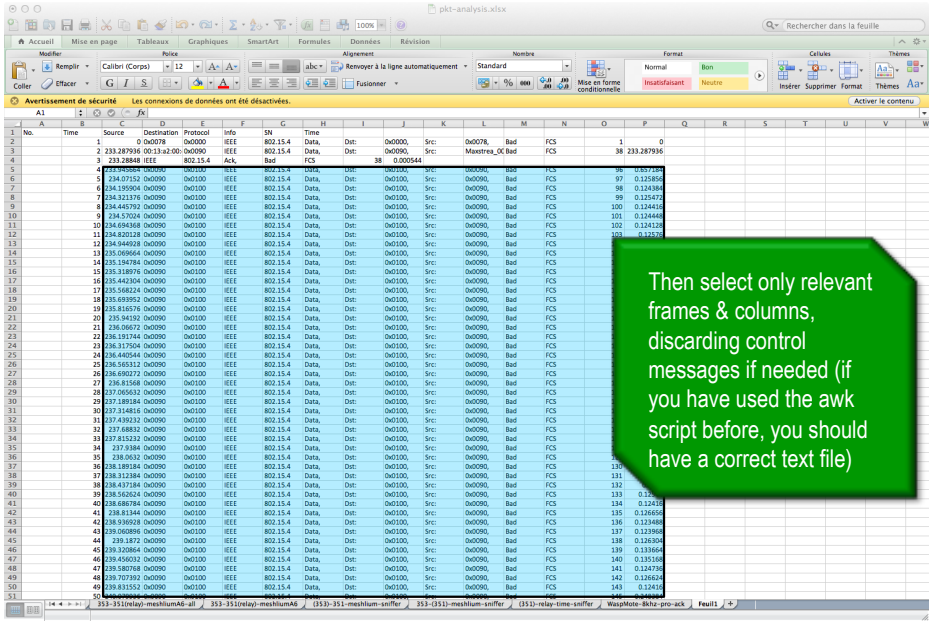
```
resources — bash — 120×24
202 261.709152 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 62
nb packet: 223 lost: 0 total lost: 24
203 261.835520 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 63
nb packet: 224 lost: 0 total lost: 24
204 261.960032 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 64
nb packet: 225 lost: 0 total lost: 24
205 262.083232 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 65
nb packet: 226 lost: 0 total lost: 24
206 262.208352 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 66
nb packet: 227 lost: 0 total lost: 24
207 262.334048 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 67
nb packet: 228 lost: 0 total lost: 24
208 262.458176 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 68
nb packet: 229 lost: 0 total lost: 24
209 262.707392 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 70
nb packet: 231 lost: 1 total lost: 25
210 263.081280 0x0090 0x0100 IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS 73
nb packet: 234 lost: 2 total lost: 27
--------------------
nb packet 234
nb lost 27
loss rate 11,5385
--------------------
MacBookProRetina-de-Congduc-Pham:resources cpham$
```

At the end, shows the total # of packet (taking into account the # of lost packets), the total # of lost packets and the final packet loss percentage

15

Use Excel to vizualize loss patterns

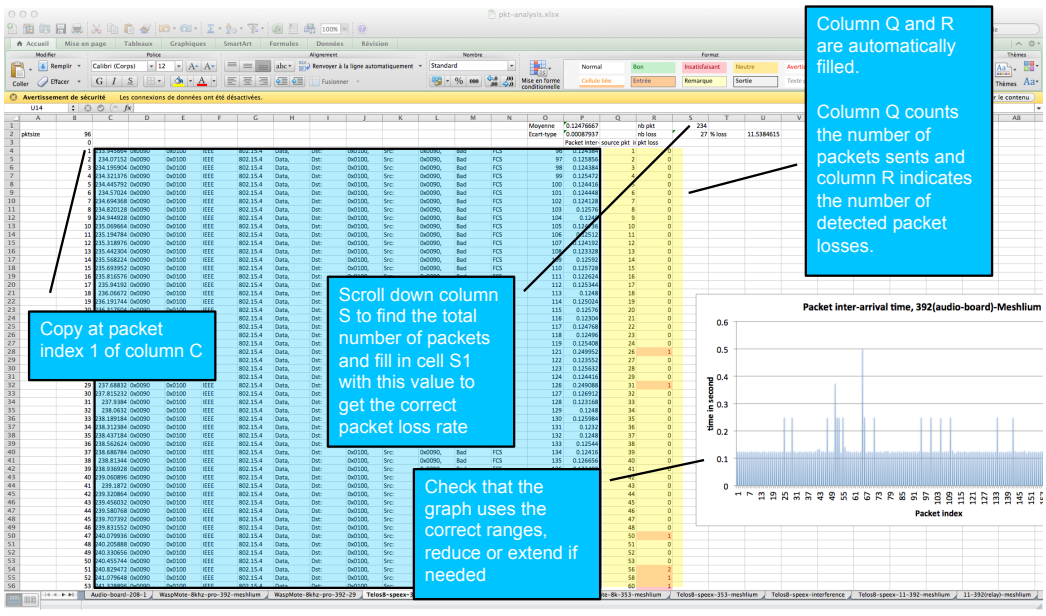
Copy/Paste the text into an Excel blank page, using text importation assistant to separate data into columns



Then select only relevant frames & columns, discarding control messages if needed (if you have used the awk script before, you should have a correct text file)

16

Copy selection into the template page



17

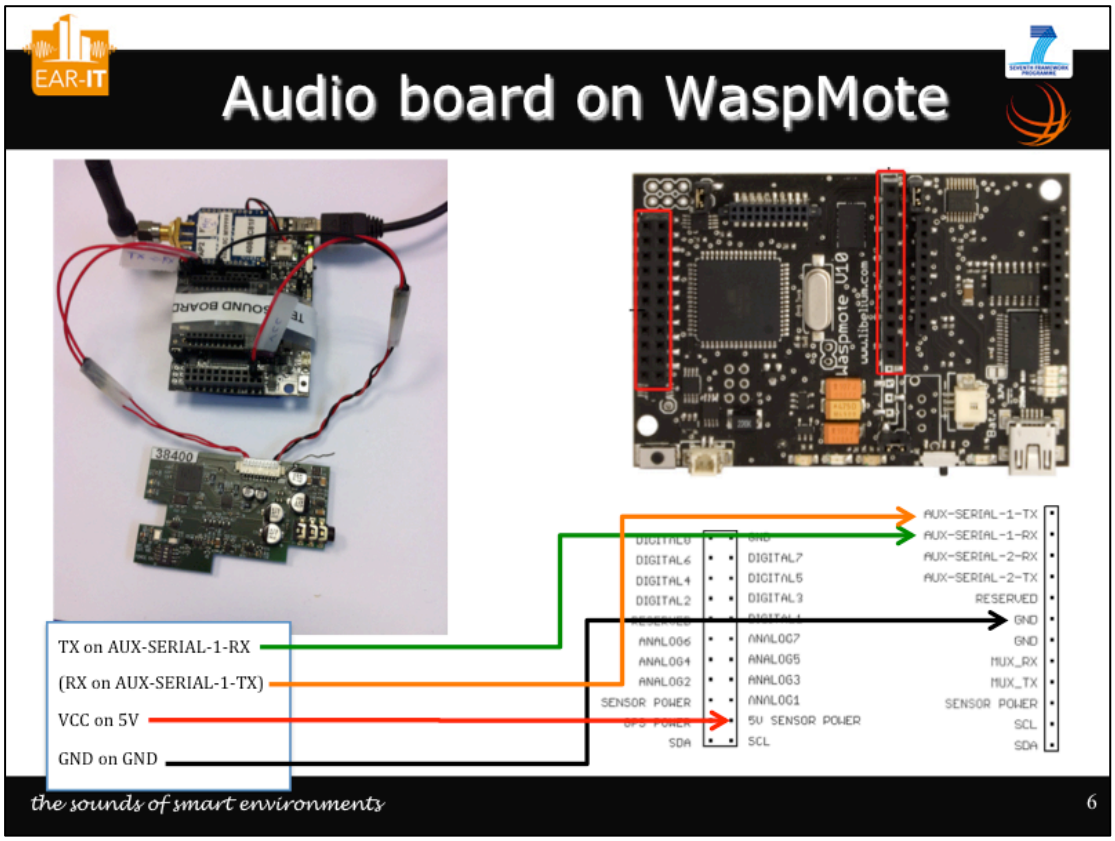
Check test-bed performances

- Refer to EAR-IT deliverable 1.2
- With 1-hop packet loss rates, check whether the value is acceptable, i.e. below 50% for raw audio and below 35% for speex audio
- Check whether the relay time of your test-bed is compatible with audio requirements, use aggregation if needed

18

ANNEX.D: Audio board on other IoT platforms

1



2