EAR-IT *the sounds of smart environment*

WP1 Acoustic Test bed Qualification

Audio test-bed description

C. Pham (EGM & LIUPPA/University of Pau) and P. Cousin (EGM)

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)

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`

Customized speex audio tools

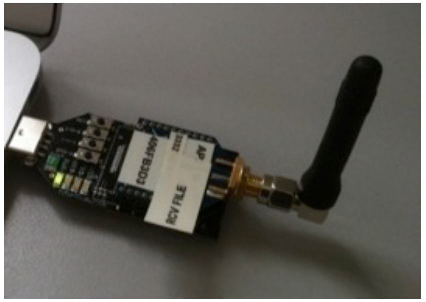
- Simple « pure » speex audio decoder without any header
  - Modified version of speex's `sampledec.c`
  - `speex_sampledec_wframing :` expects framing bytes
  - `speex_sampledec_nframing :` no framing bytes

- 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`

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)

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
```

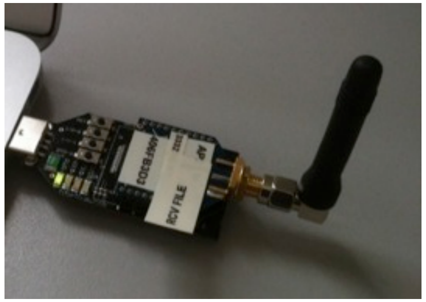
SerialToStdout.py

- 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

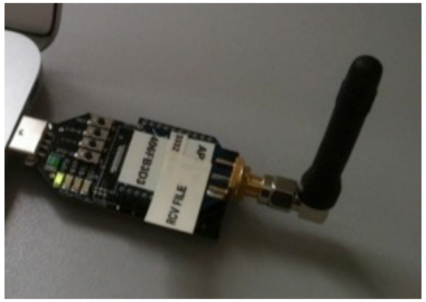
XBeeSendCmd



- 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)
```

XBeeSendFile

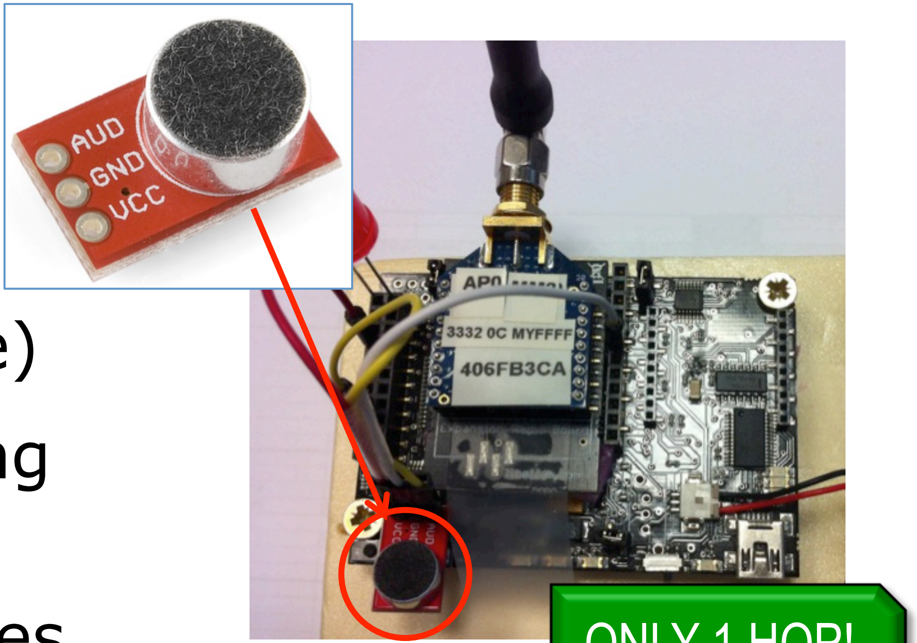


- `XBeeSendFile`
  - Main target is 802.15.4 XBee-based gateway
  - Send binary files with Xbee with controlled timing
  - Can use any packet size between 1 and 100 bytes
  - Can insert framing bytes, can introduce packet losses

```
USAGE:      ./XBeeSendFile -baud baudrate -p dev -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:      -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
```

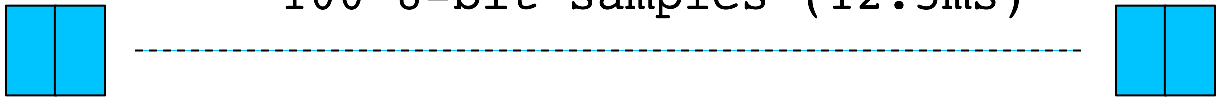
WaspMote+XBee in raw mode

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

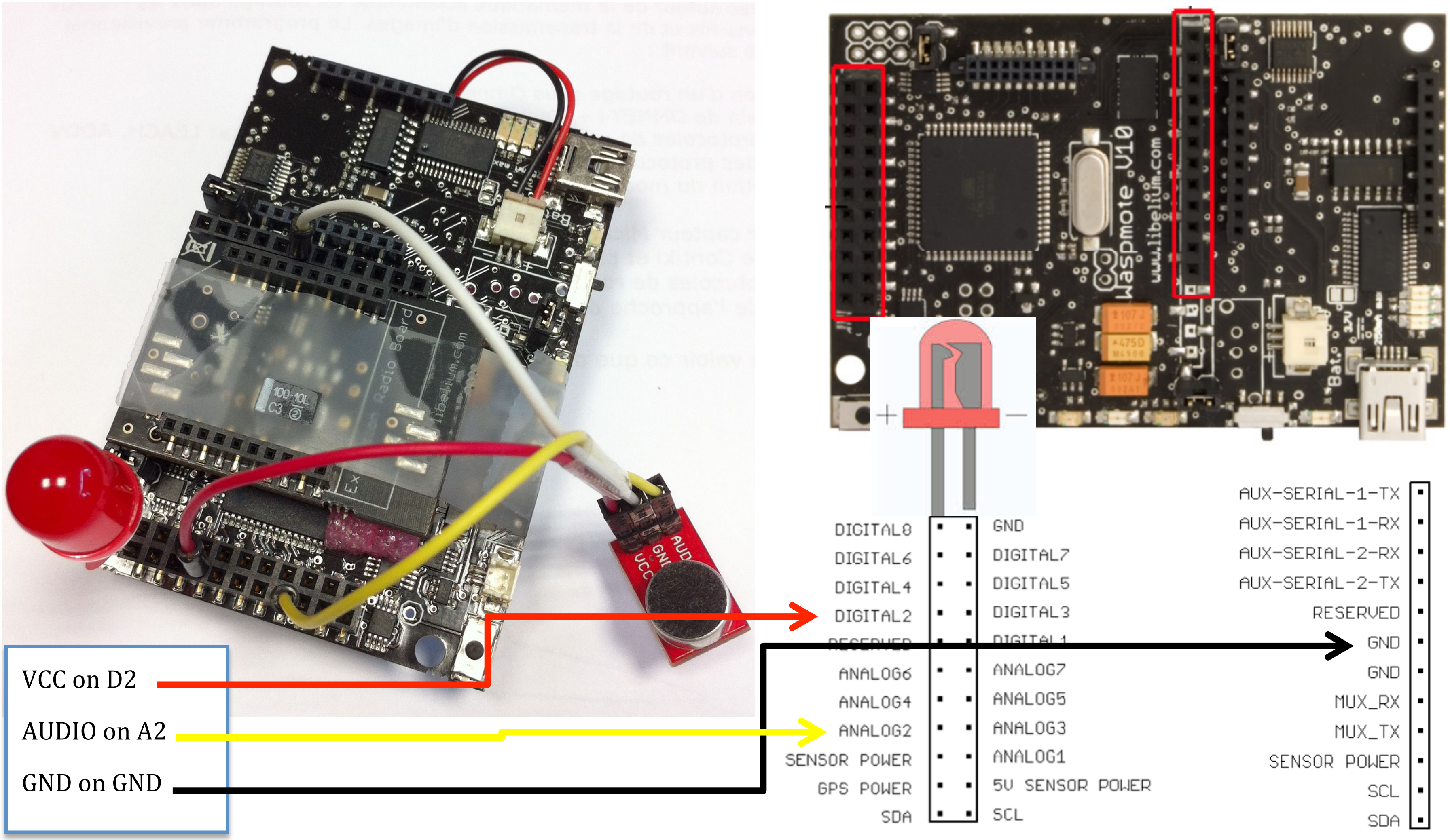


ONLY 1 HOP!

100 8-bit samples (12.5ms)



Details of pin connection



VCC on D2

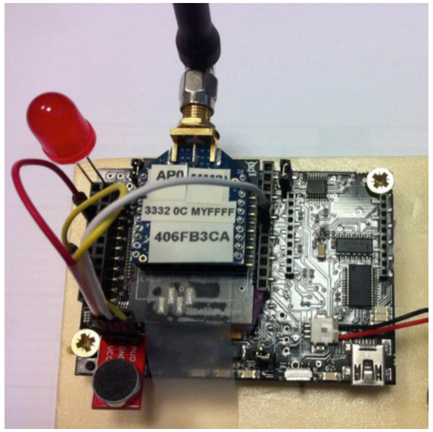
AUDIO on A2

GND on GND

DIGITAL8	GND
DIGITAL6	DIGITAL7
DIGITAL4	DIGITAL5
DIGITAL2	DIGITAL3
RESERVED	DIGITAL1
ANALOG6	ANALOG7
ANALOG4	ANALOG5
ANALOG2	ANALOG3
SENSOR POWER	ANALOG1
GPS POWER	5V SENSOR POWER
SDA	SCL

AUX-SERIAL-1-TX
AUX-SERIAL-1-RX
AUX-SERIAL-2-RX
AUX-SERIAL-2-TX
RESERVED
GND
GND
MUX_RX
MUX_TX
SENSOR POWER
SCL
SDA

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

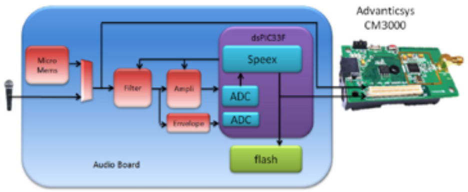
```
> python SerialToStdout | play --buffer 50 -t raw —r 4000 -u -1 —
```

XBee gateway in pkt mode (AP2)

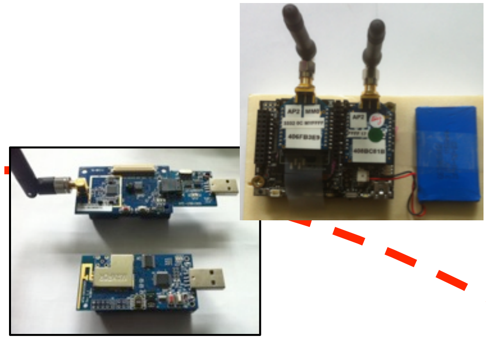
- The receiving XBee module may need to be in packet mode (AP2) due to deployment constraints
- Adds overhead of XBee API frame decoding: 8KHz sampling may be not supported

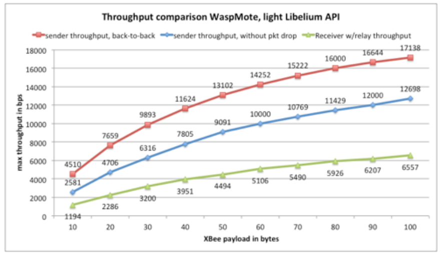
```
4KHz sampling
> XBeeReceive -baud 38400 –stream dumb.dat | play --buffer 50 -t raw –r 4000 -u -1 –


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

Multi-hop audio constraints

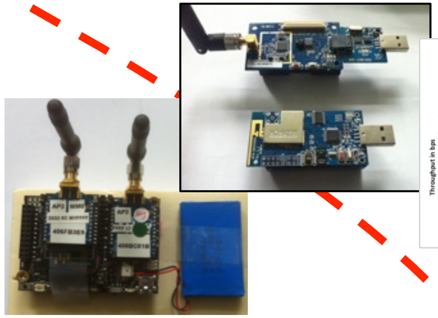


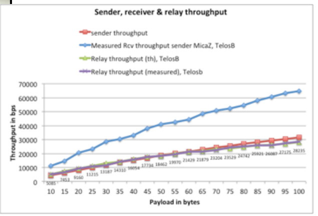
NEED AUDIO ENCODING
TO REDUCE AUDIO DATA SIZE



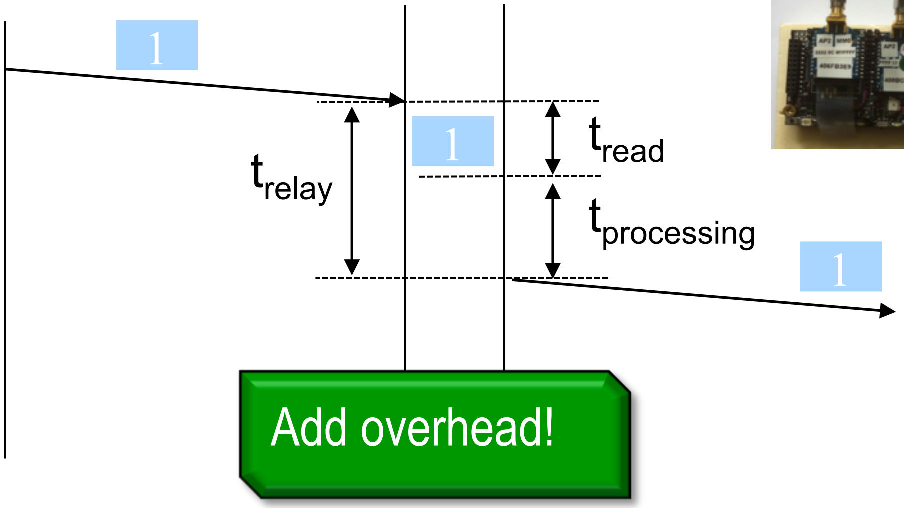


RELAY

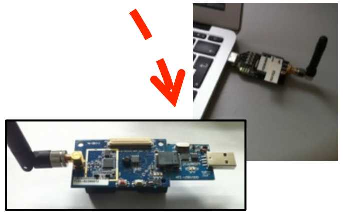




RELAY



Add overhead!

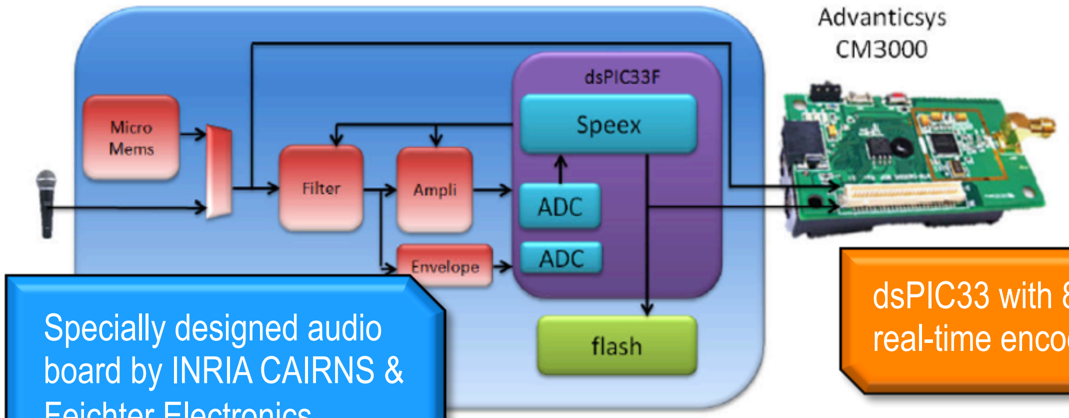


DECODE & PLAY
RECEIVED AUDIO

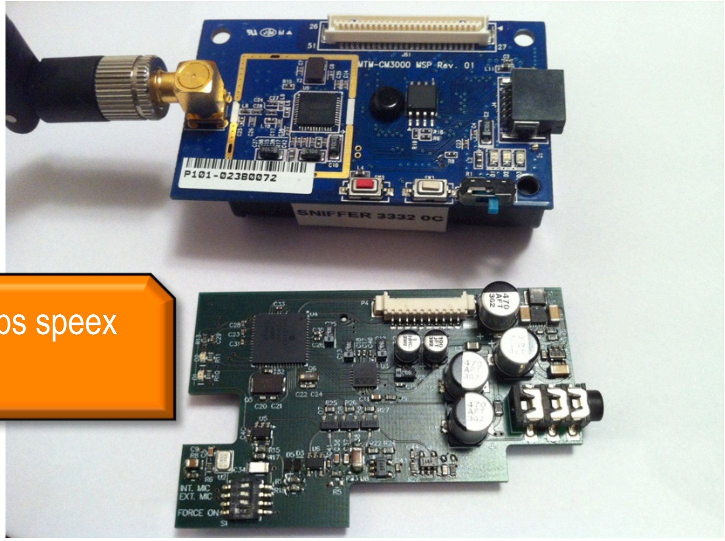
the sounds of smart environments

Multi-hop audio solution

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



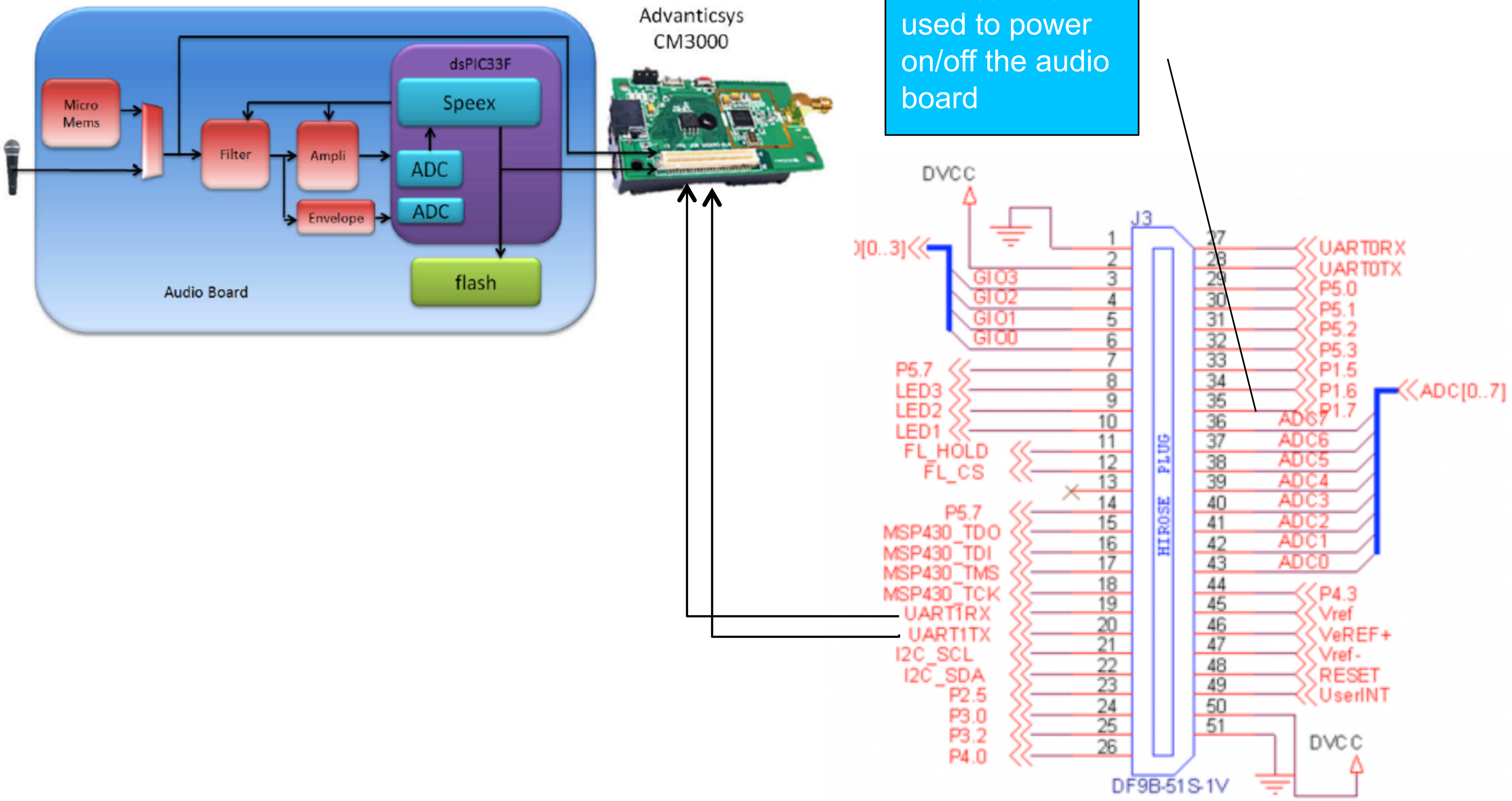
Specially designed audio board by INRIA CAIRNS & Feichter Electronics



dsPIC33 with 8kbps speex real-time encoder

- Allows for multi-hop, encoded audio streaming scenarios

Details of pin connection



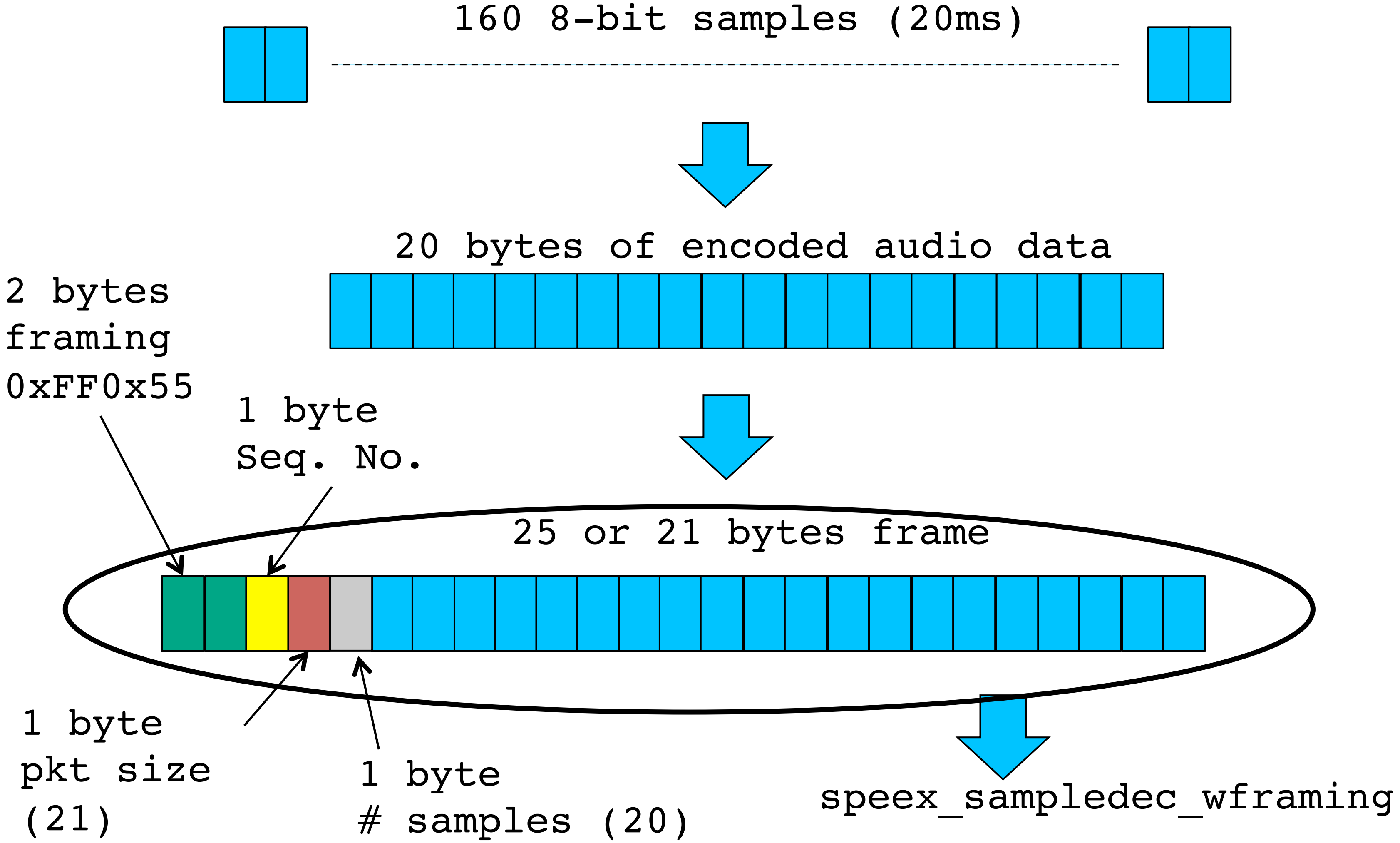
P1.7 can be used to power on/off the audio board

the sounds of smart environments

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`)

speex at 8kbps



160 8-bit samples (20ms)

20 bytes of encoded audio data

2 bytes framing 0xFF0x55

1 byte Seq. No.

25 or 21 bytes frame

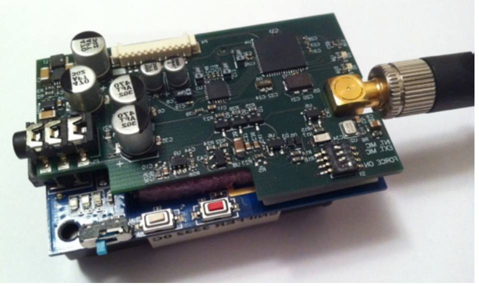
1 byte pkt size (21)

1 byte # samples (20)

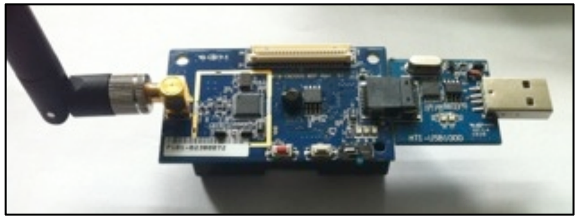
speex_sampledec_wframing

the sounds of smart environments

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 -
```

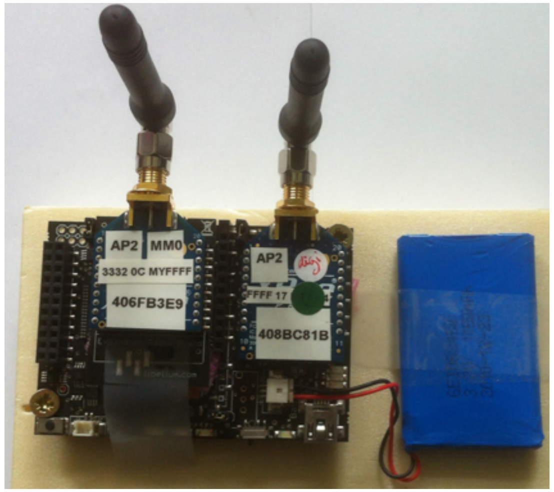
With XBee GW in AP0 mode

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

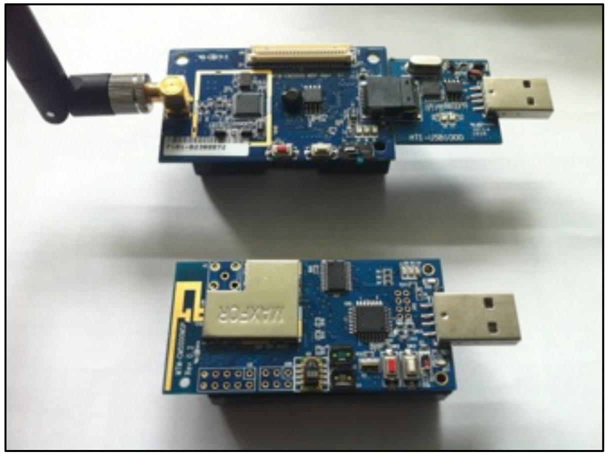
With XBee GW in AP2 mode (pkt mode)

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

Relay nodes



Libelium WaspMote



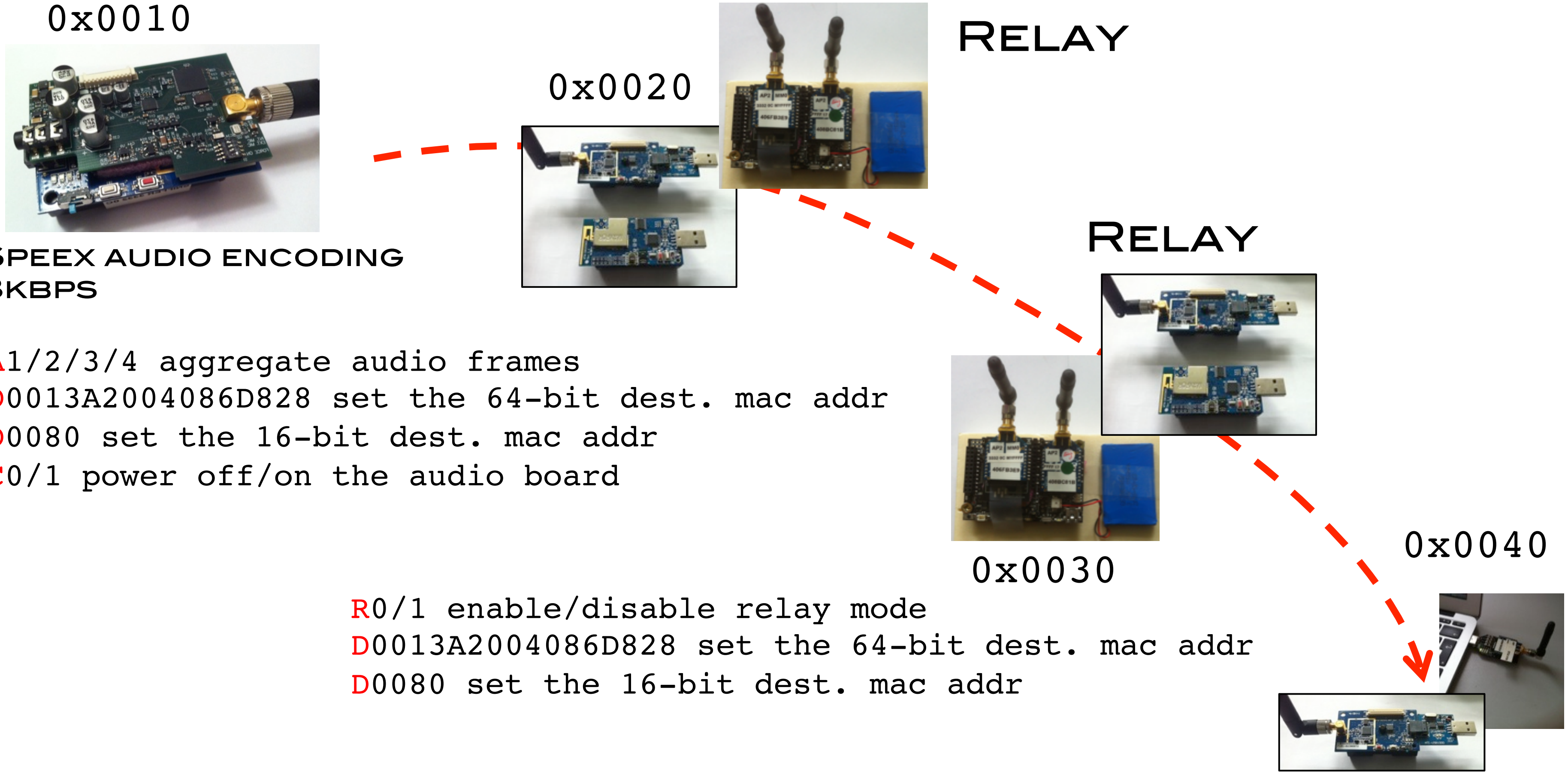
AdvanticSys CM5000, CM3000

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
```

Multi-hop test-bed w/audio board



0x0010

Speex audio encoding
8kbps

```
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
```

0x0020

Relay

Relay

0x0030

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

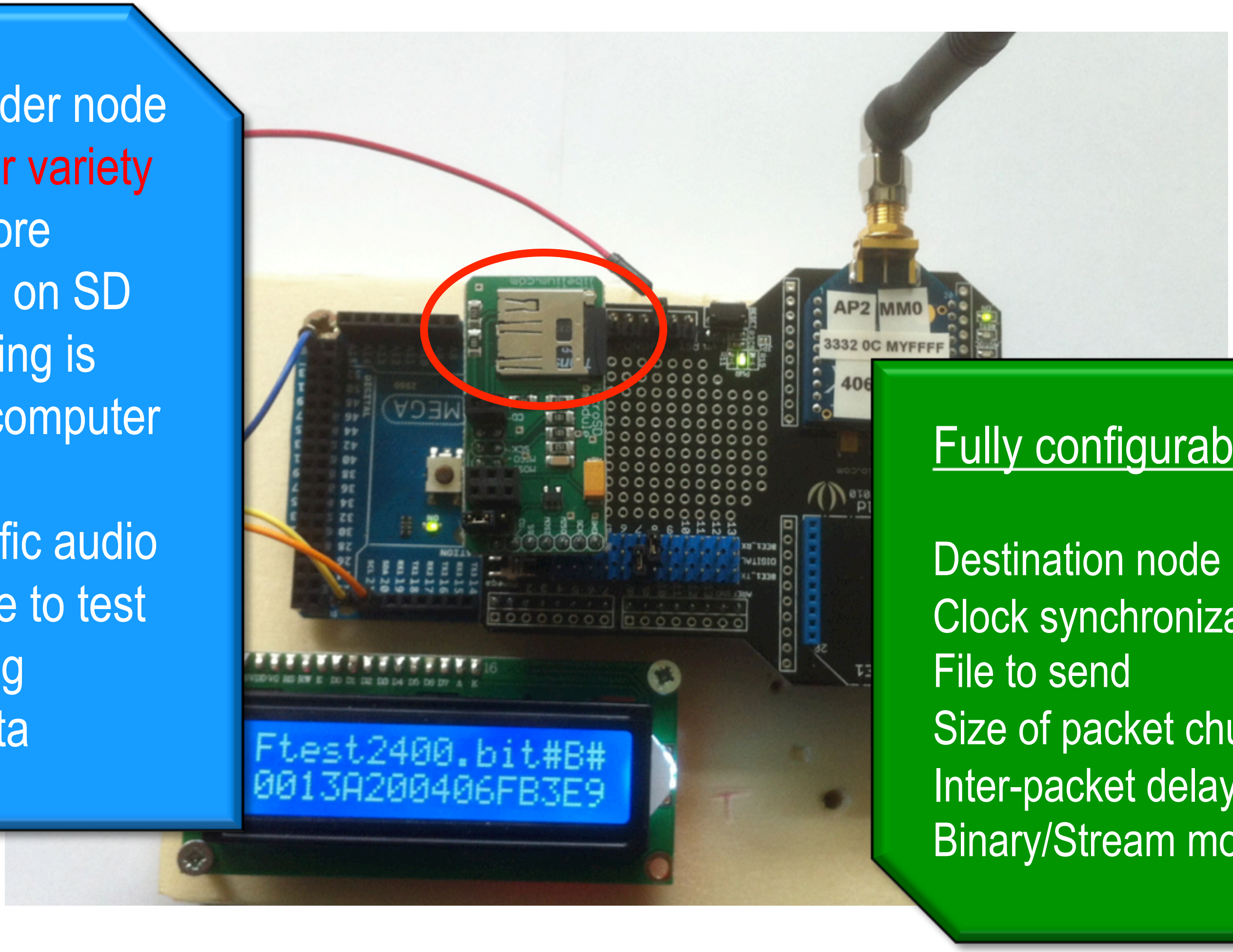
0x0040

Decode & Play
Received audio

Generic & controlled sender

Use a generic sender node to test with a larger variety of audio codec: store encoded audio file on SD card. Audio encoding is done on desktop computer

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

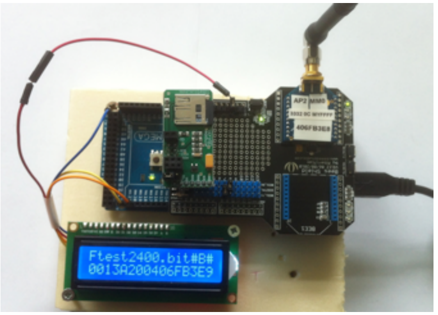


Fully configurable:

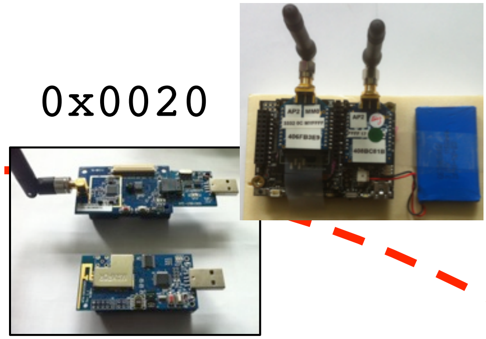
Destination node
Clock synchronization
File to send
Size of packet chunk
Inter-packet delay
Binary/Stream mode

Multi-hop test-bed w/generic sender

0x0010

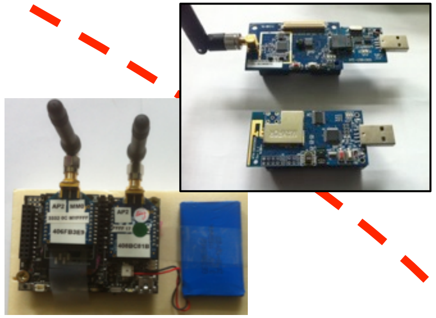


0x0020



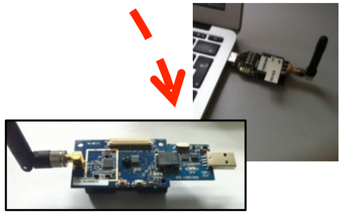
Relay

Relay



0x0030

0x0040



Decode & Play
Received audio

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#`

codec2/speex with generic sender

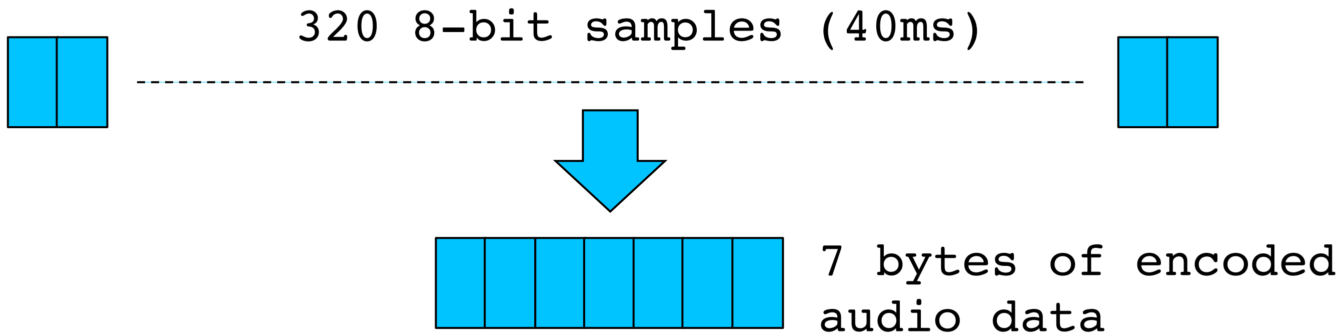
- Use codec2/speex encoding software to produce encoded audio file
- Store encoded audio file (.bit/.spx) on SD card
- Configure the generic sender for sending the encoded audio file
  - Define packet size
  - Determine inter-packet send time
- Receive the encoded audio stream, decode the data and determine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

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

Codec2 encoding

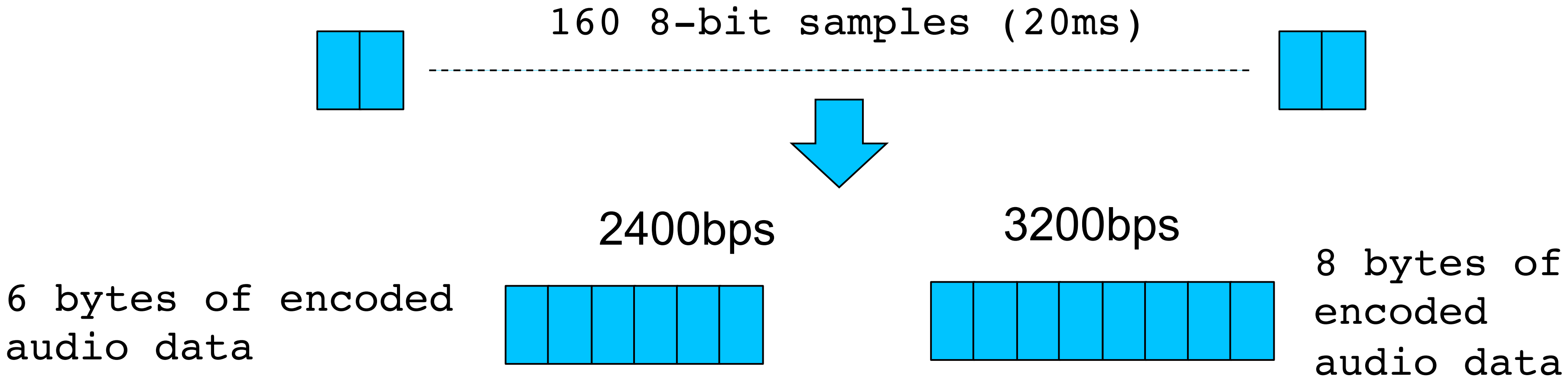
at 1400bps



320 8-bit samples (40ms)

7 bytes of encoded audio data

at 2400bps & 3200bps



160 8-bit samples (20ms)

2400bps

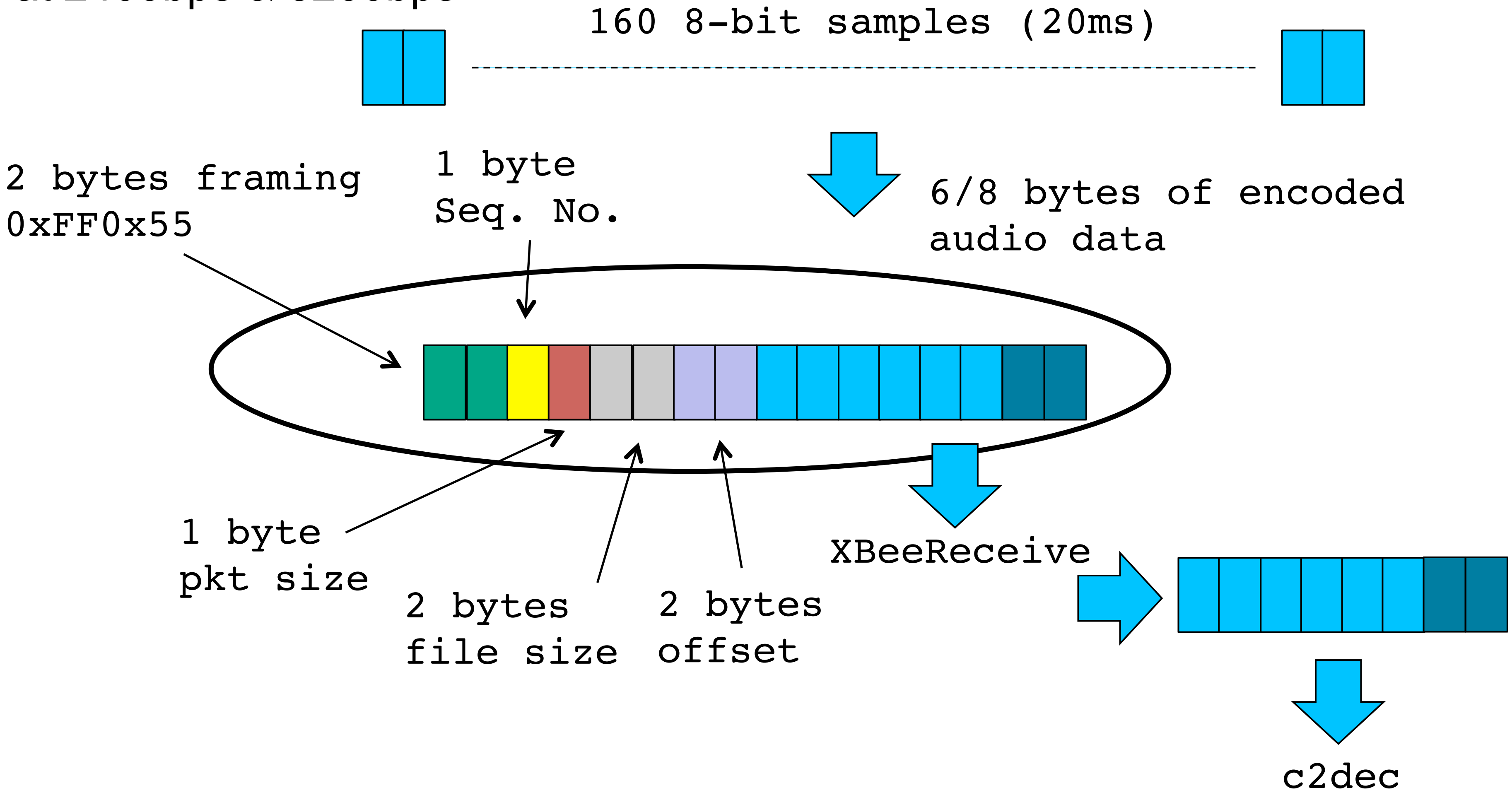
3200bps

6 bytes of encoded audio data

8 bytes of encoded audio data

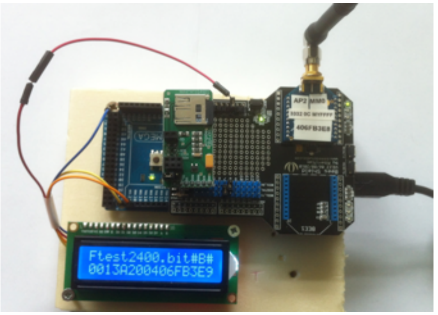
Codec2 at 2400bps & 3200

at 2400bps & 3200bps

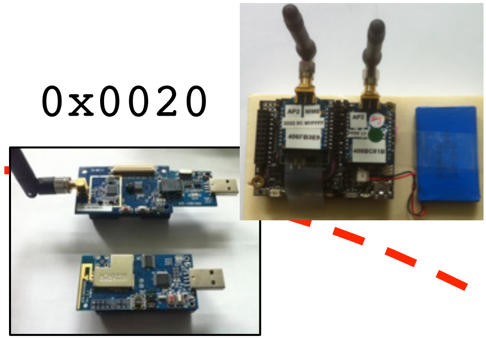


Multi-hop tests with codec2

0x0010

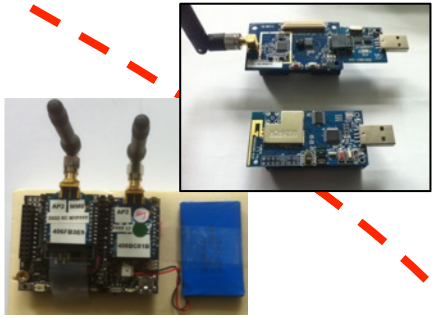


0x0020



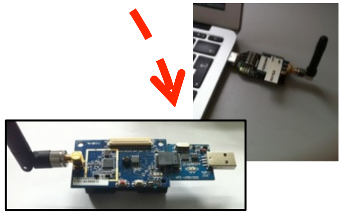
Relay

Relay



0x0030

0x0040



Decode & Play
Received Audio

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 –
```

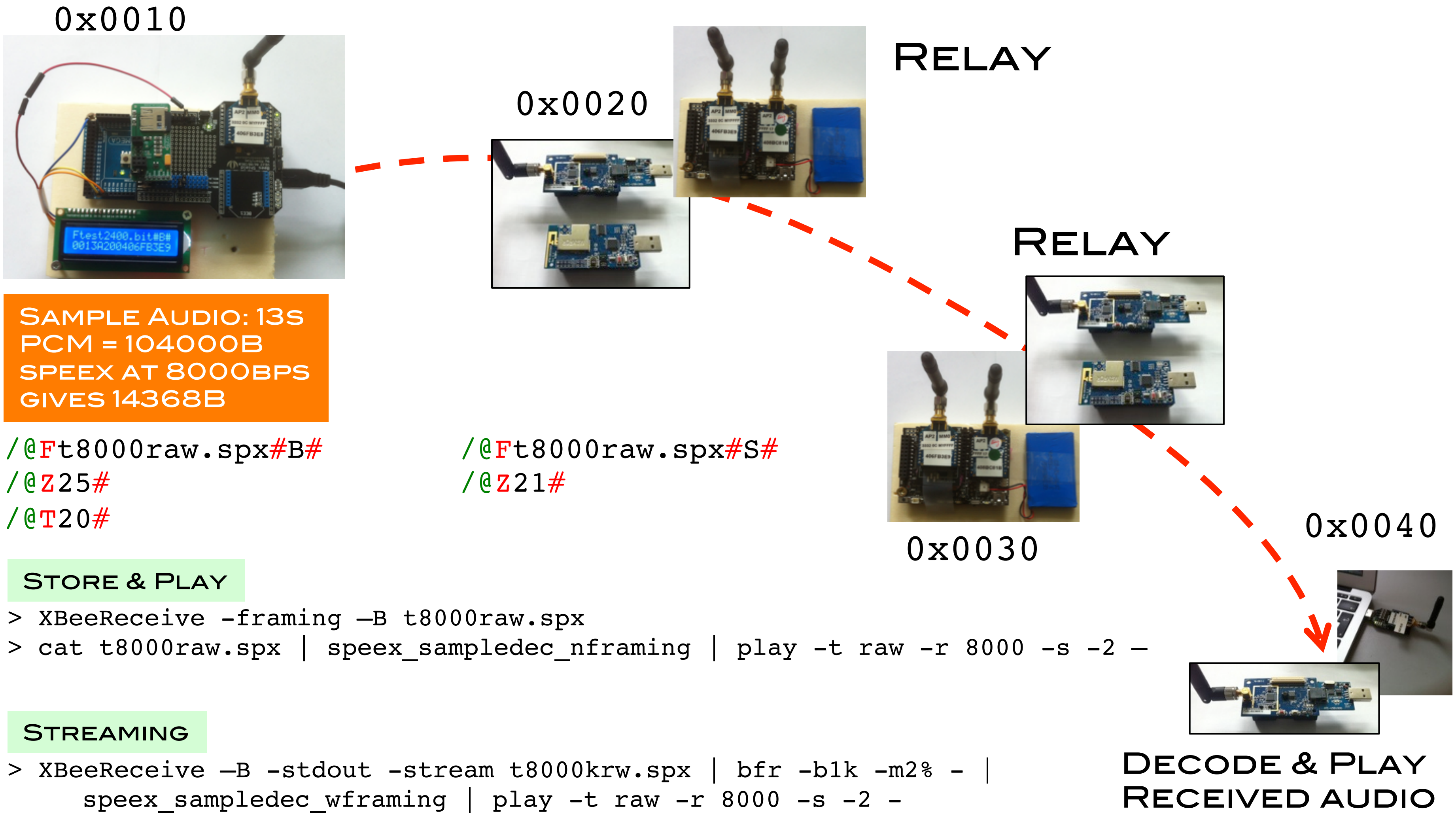
Streaming

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

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

Multi-hop tests with speex



Sample Audio: 13s
PCM = 104000B
speex at 8000bps
gives 14368B

```
/@Ft8000raw.spx#B#
/@Z25#
/@T20#
```

```
/@Ft8000raw.spx#S#
/@Z21#
```

Store & Play

```
> XBeeReceive -framing —B t8000raw.spx
> cat t8000raw.spx | speex_sampledec_nframing | play -t raw -r 8000 -s -2 –
```

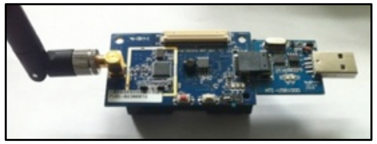
Streaming

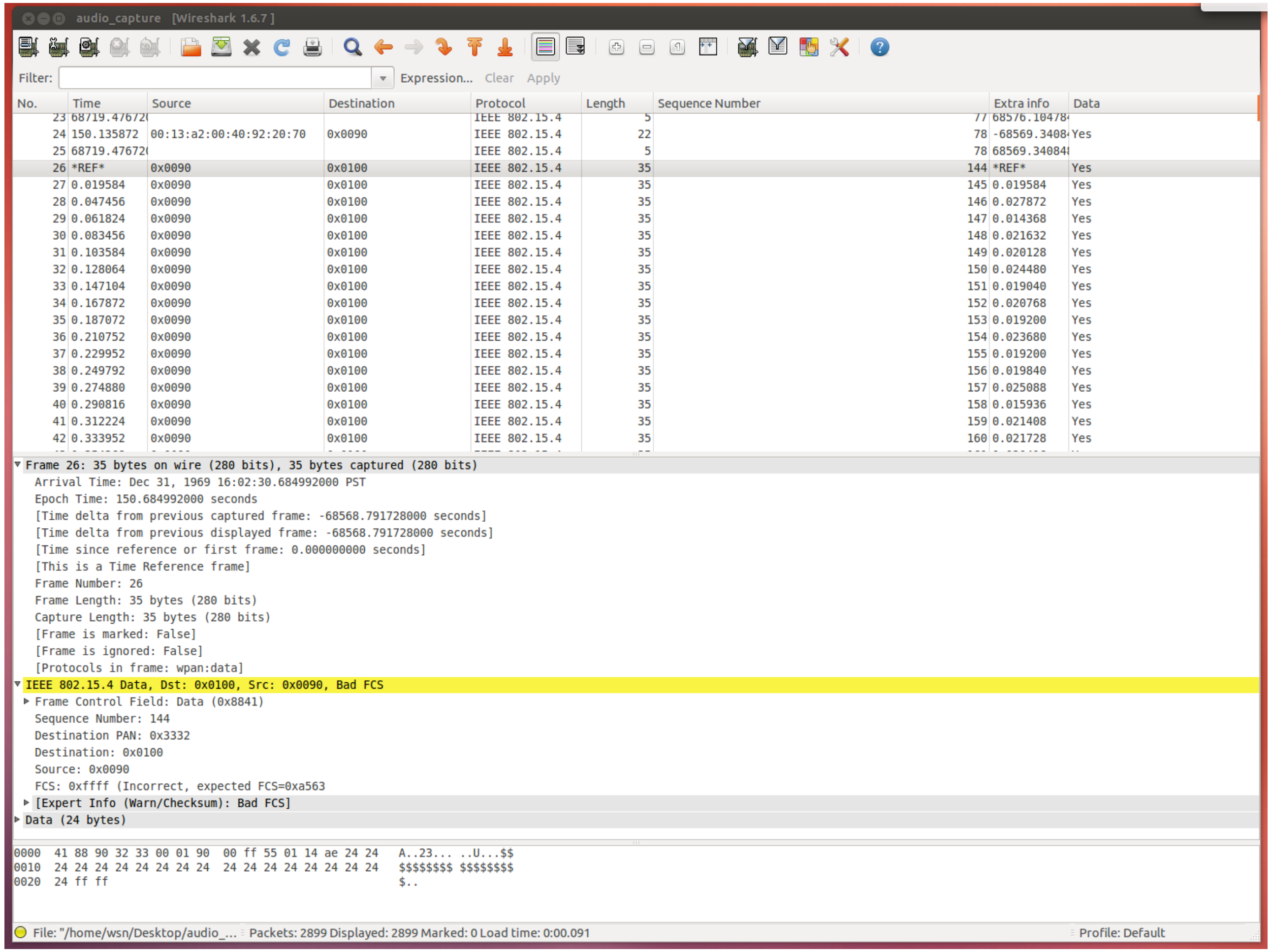
```
> XBeeReceive —B -stdout -stream t8000krw.spx | bfr -b1k -m2% - |
     speex_sampledec_wframing | play -t raw -r 8000 -s -2 -
```

Frame analysis

- Use `wireshark` as frame analysis tool
- AdvanticSys TelosB mote as promiscuous sniffer mote, connected to `wireshark` to display captured frames
- Frame reception time can be visualized for statistic collection
  - Transmission latencies
  - Frame jitter

wireshark frame capture





No.	Time	Source	Destination	Protocol	Length	Sequence Number	Extra info	Data
23	68719.47672(			IEEE 802.15.4	5	77	68576.10478	
24	150.135872	00:13:a2:00:40:92:20:70	0x0090	IEEE 802.15.4	22	78	-68569.3408	Yes
25	68719.47672(			IEEE 802.15.4	5	78	68569.34084	
26	*REF*	0x0090	0x0100	IEEE 802.15.4	35	144	*REF*	Yes
27	0.019584	0x0090	0x0100	IEEE 802.15.4	35	145	0.019584	Yes
28	0.047456	0x0090	0x0100	IEEE 802.15.4	35	146	0.027872	Yes
29	0.061824	0x0090	0x0100	IEEE 802.15.4	35	147	0.014368	Yes
30	0.083456	0x0090	0x0100	IEEE 802.15.4	35	148	0.021632	Yes
31	0.103584	0x0090	0x0100	IEEE 802.15.4	35	149	0.020128	Yes
32	0.128064	0x0090	0x0100	IEEE 802.15.4	35	150	0.024480	Yes
33	0.147104	0x0090	0x0100	IEEE 802.15.4	35	151	0.019040	Yes
34	0.167872	0x0090	0x0100	IEEE 802.15.4	35	152	0.020768	Yes
35	0.187072	0x0090	0x0100	IEEE 802.15.4	35	153	0.019200	Yes
36	0.210752	0x0090	0x0100	IEEE 802.15.4	35	154	0.023680	Yes
37	0.229952	0x0090	0x0100	IEEE 802.15.4	35	155	0.019200	Yes
38	0.249792	0x0090	0x0100	IEEE 802.15.4	35	156	0.019840	Yes
39	0.274880	0x0090	0x0100	IEEE 802.15.4	35	157	0.025088	Yes
40	0.290816	0x0090	0x0100	IEEE 802.15.4	35	158	0.015936	Yes
41	0.312224	0x0090	0x0100	IEEE 802.15.4	35	159	0.021408	Yes
42	0.333952	0x0090	0x0100	IEEE 802.15.4	35	160	0.021728	Yes

```
▼ Frame 26: 35 bytes on wire (280 bits), 35 bytes captured (280 bits)
    Arrival Time: Dec 31, 1969 16:02:30.684992000 PST
    Epoch Time: 150.684992000 seconds
    [Time delta from previous captured frame: -68568.791728000 seconds]
    [Time delta from previous displayed frame: -68568.791728000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    [This is a Time Reference frame]
    Frame Number: 26
    Frame Length: 35 bytes (280 bits)
    Capture Length: 35 bytes (280 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: wpan:data]
▼ IEEE 802.15.4 Data, Dst: 0x0100, Src: 0x0090, Bad FCS
  ▶ Frame Control Field: Data (0x8841)
    Sequence Number: 144
    Destination PAN: 0x3332
    Destination: 0x0100
    Source: 0x0090
    FCS: 0xffff (Incorrect, expected FCS=0xa563
  ▶ [Expert Info (Warn/Checksum): Bad FCS]
▶ Data (24 bytes)
```

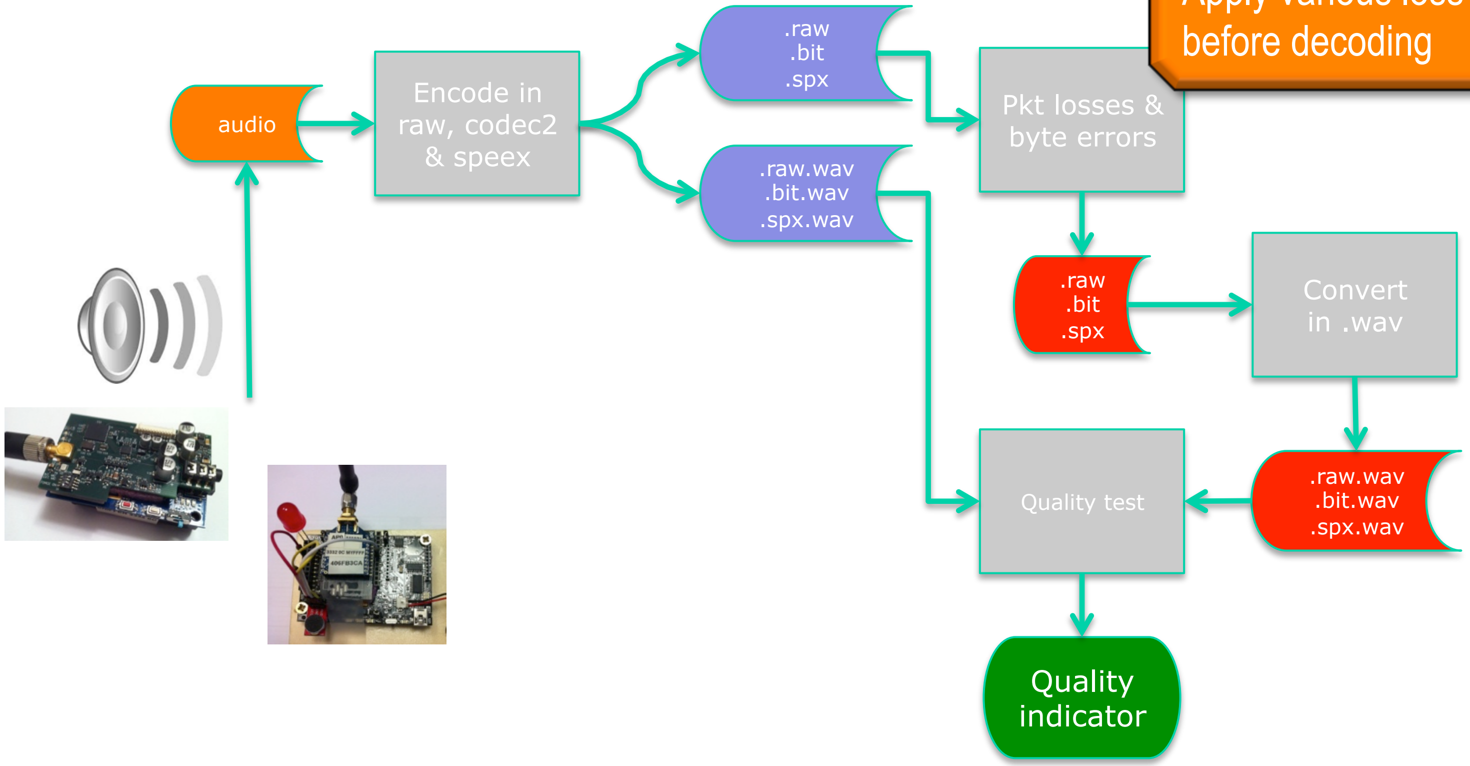
```
0000  41 88 90 32 33 00 01 90  00 ff 55 01 14 ae 24 24   A..23... ..U...$$
0010  24 24 24 24 24 24 24 24  24 24 24 24 24 24 24 24   $$$$$$$$ $$$$$$$$
0020  24 ff ff                                           $..
```

File: "/home/wsn/Desktop/audio_... Packets: 2899 Displayed: 2899 Marked: 0 Load time: 0:00.091 Profile: Default

Sensitivity of codecs

LAB TESTS

Apply various loss patterns before decoding



audio

Encode in raw, codec2 & speex

.raw .bit .spx

.raw.wav .bit.wav .spx.wav

Pkt losses & byte errors

.raw .bit .spx

Convert in .wav

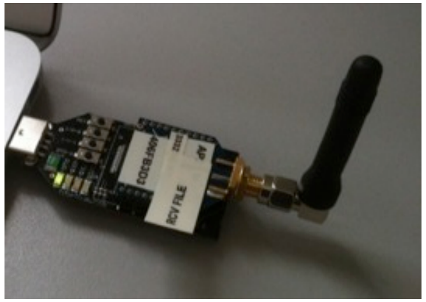
.raw.wav .bit.wav .spx.wav

Quality test

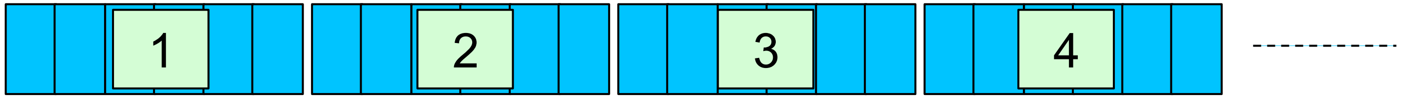
Quality indicator

Apply packet loss rate

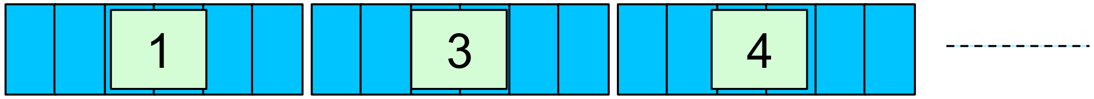
- Use XBeeSendFile to control
  - Timing between packet sending
  - Packet loss probability



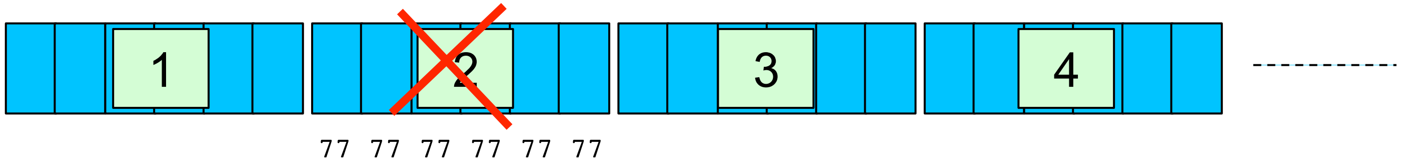
`Codec2 2400bps, series of 6-byte encoded audio packets`



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



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


1
2
3
4
----
77 77 77 77 77 77