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

C. Pham, V. Lecuire, J.M. Moureaux

IEEE WiMob 2013

October 7th-9th, 2013, Lyon

Prof. Congduc Pham
http://www.univ-pau.fr/~cpham
Université de Pau, France

Search & Rescue, Situation awareness

Imote2

Multimedia board

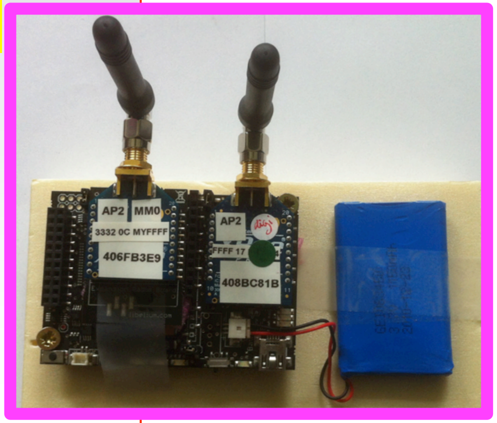
FDVision
Space Imaging 12/27/03

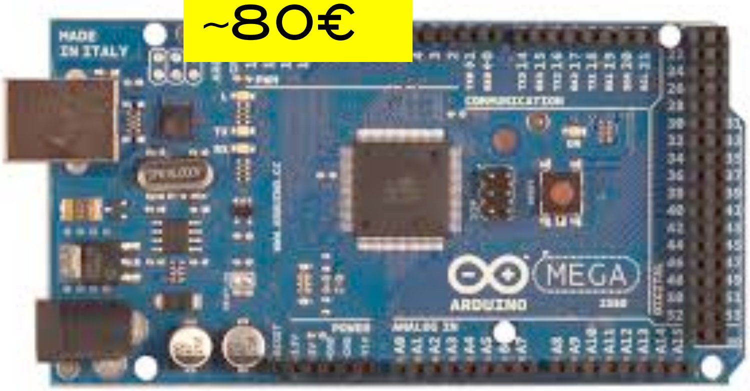
Low-cost sensors

- ❑ ATmega1281 microcontroller
- ❑ 8MHz, 4K RAM & 2G SD card.
- ❑ 2.4GHz IEEE 802.15.4 XBee

Cost: ~100€

Libelium WaspMote



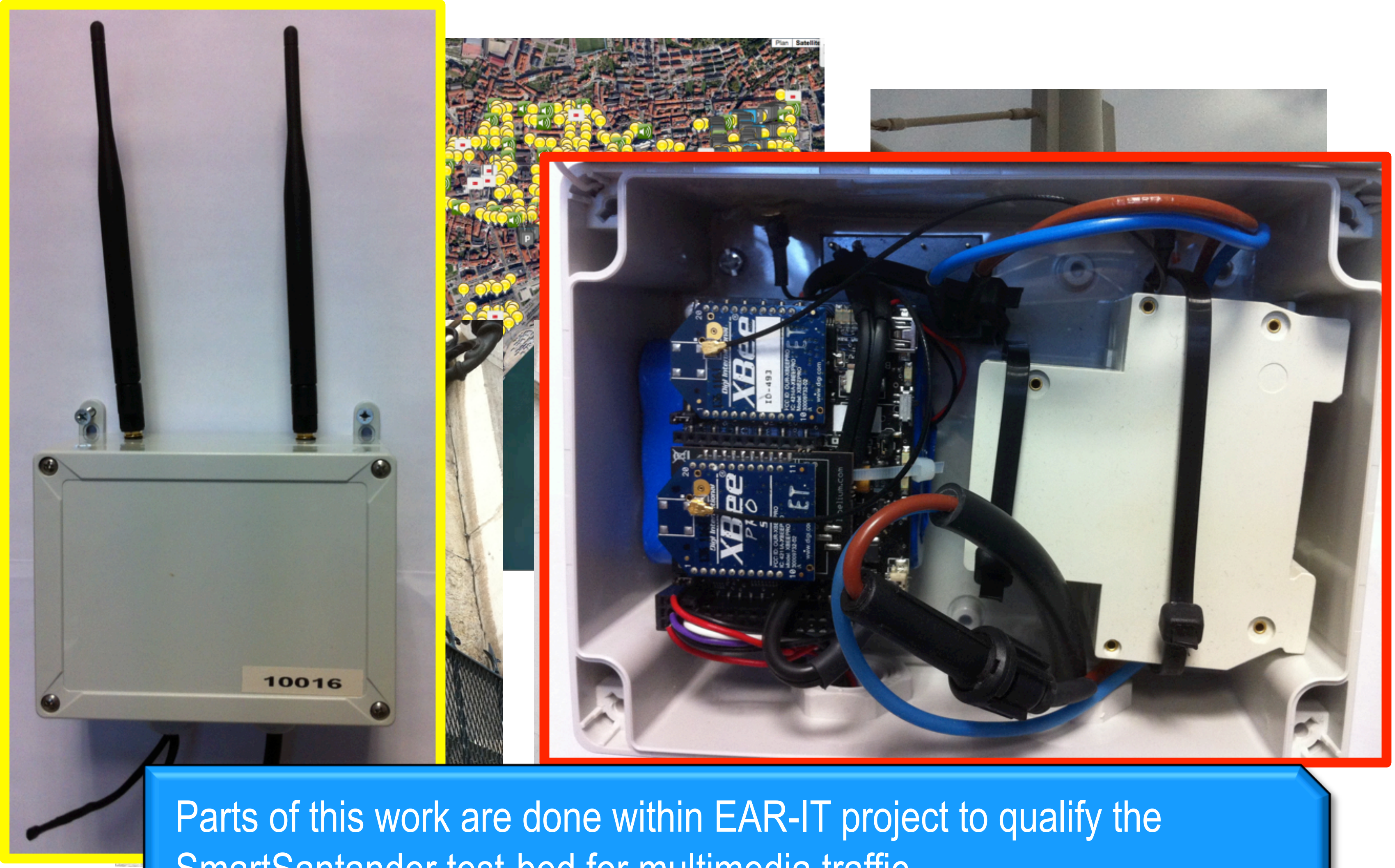


Cost: ~80€

Arduino Mega2560

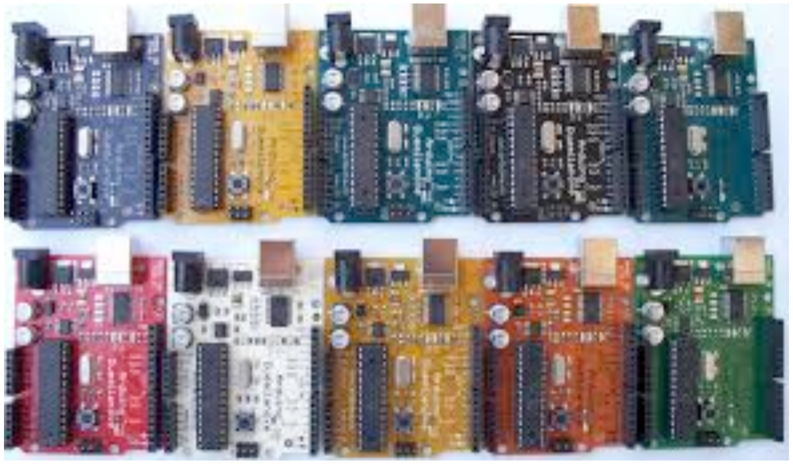
- ❑ ATmega2560 microcontroller
- ❑ 16MHz, 8K RAM
- ❑ 2.4GHz IEEE 802.15.4 XBee with XBee shield

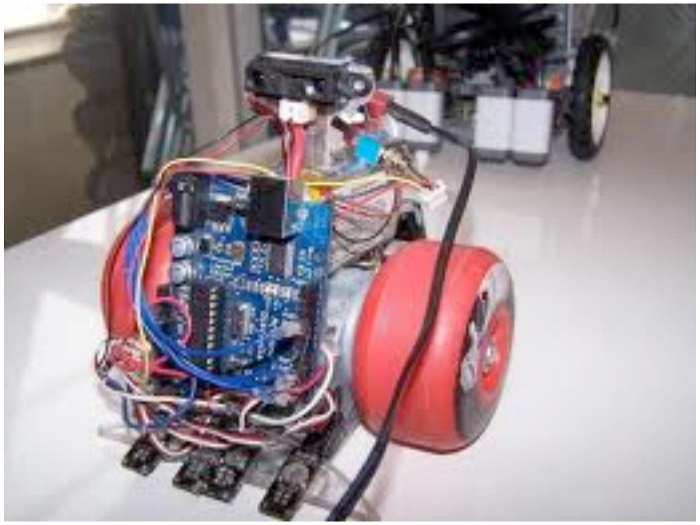
Libelium WaspMote in SmartSantander

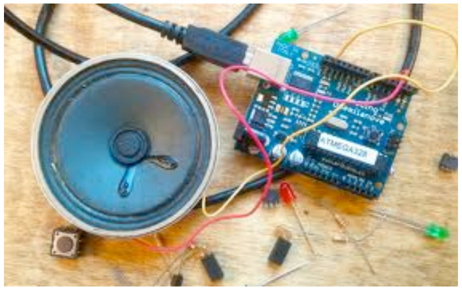


Parts of this work are done within EAR-IT project to qualify the SmartSantander test-bed for multimedia traffic

Arduino: the hobbyist dev. platform

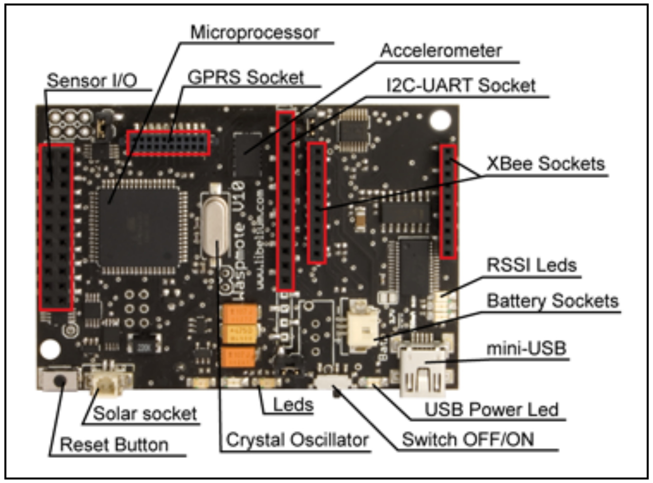






Sensor architecture

Libelium WaspMote



Arduino Mega2560

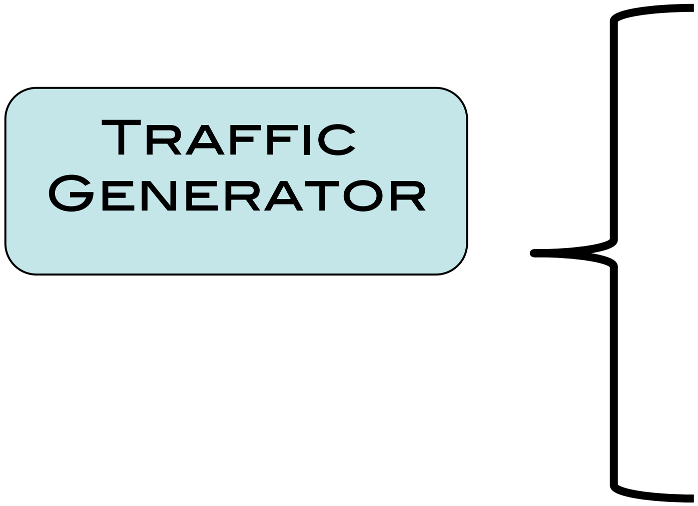
UART-based connection to micro-controller

Default speed is usually 38400 bauds

Higher baud rate are possible but…

Xbee 802.15.4

Sending performances



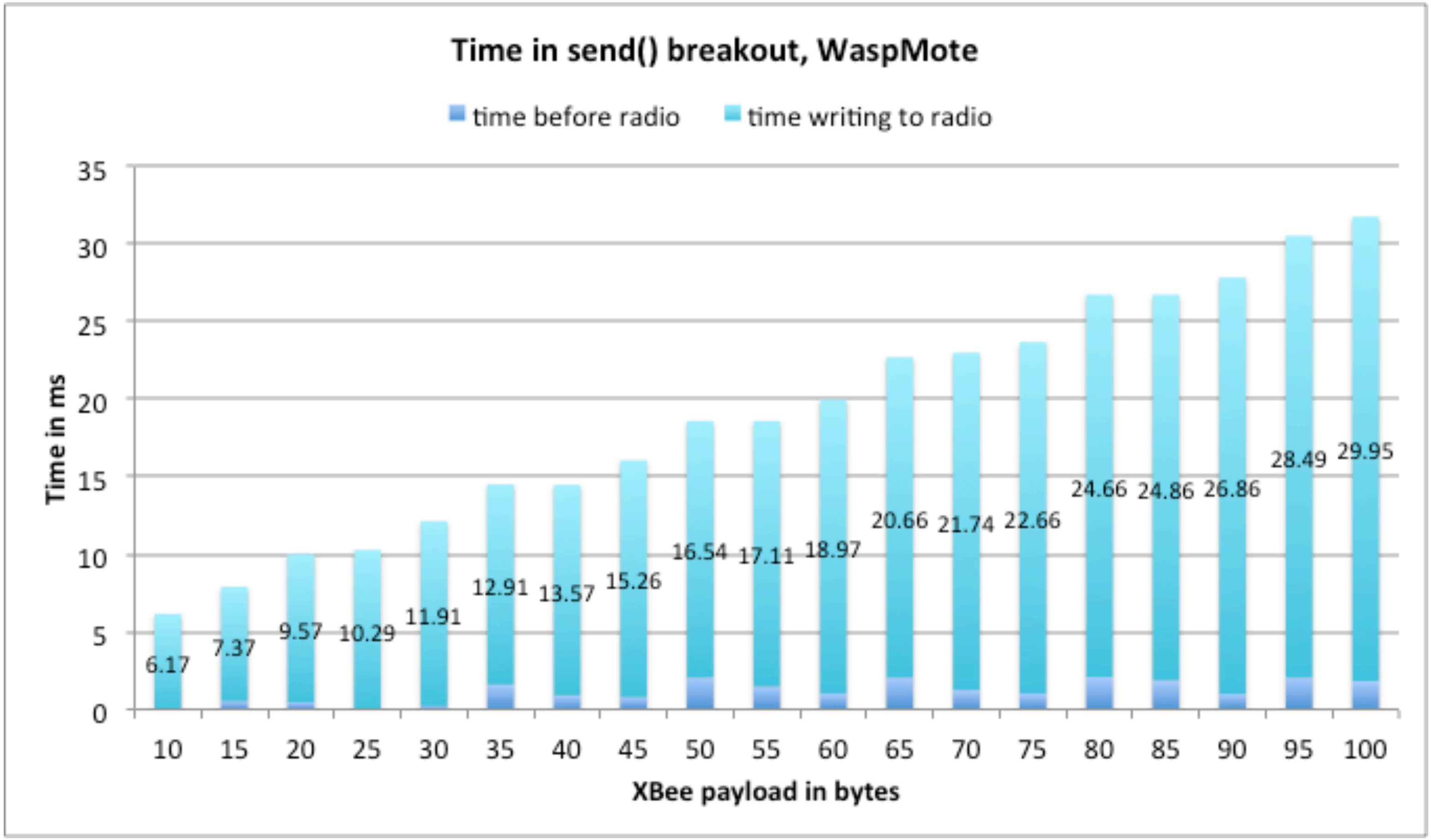
Traffic Generator

```
void loop() {
        T0;
        L0=T0;
        ...
        T1;
        send(buf);
        T2;
        ...
}
```

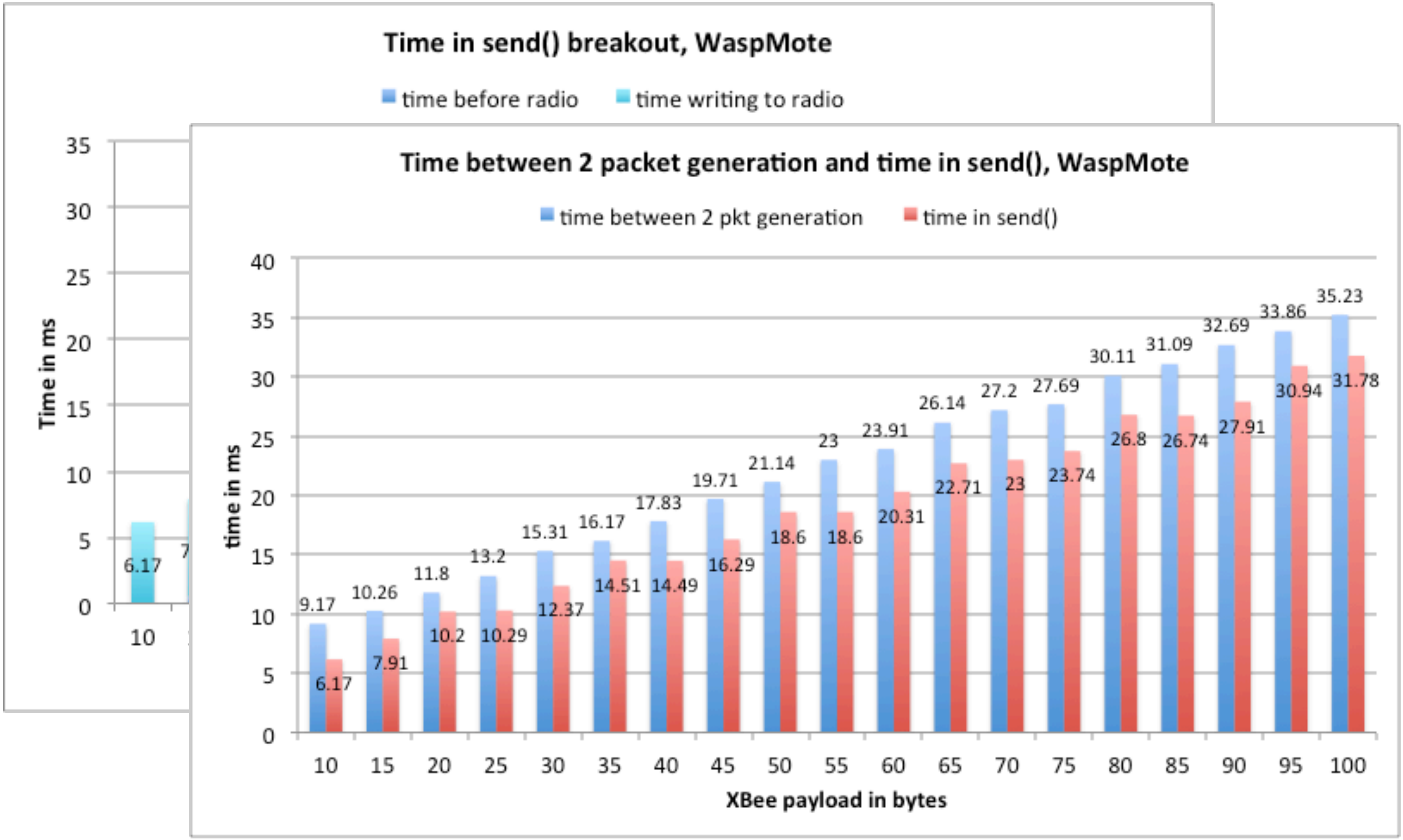
Measure the time in various part of API `send()` when possible.

« Time in send() » is T2-T1
« Time between 2 pkt generation » is T0-L0
Time resolution is millisecond
Minimum data manipulation

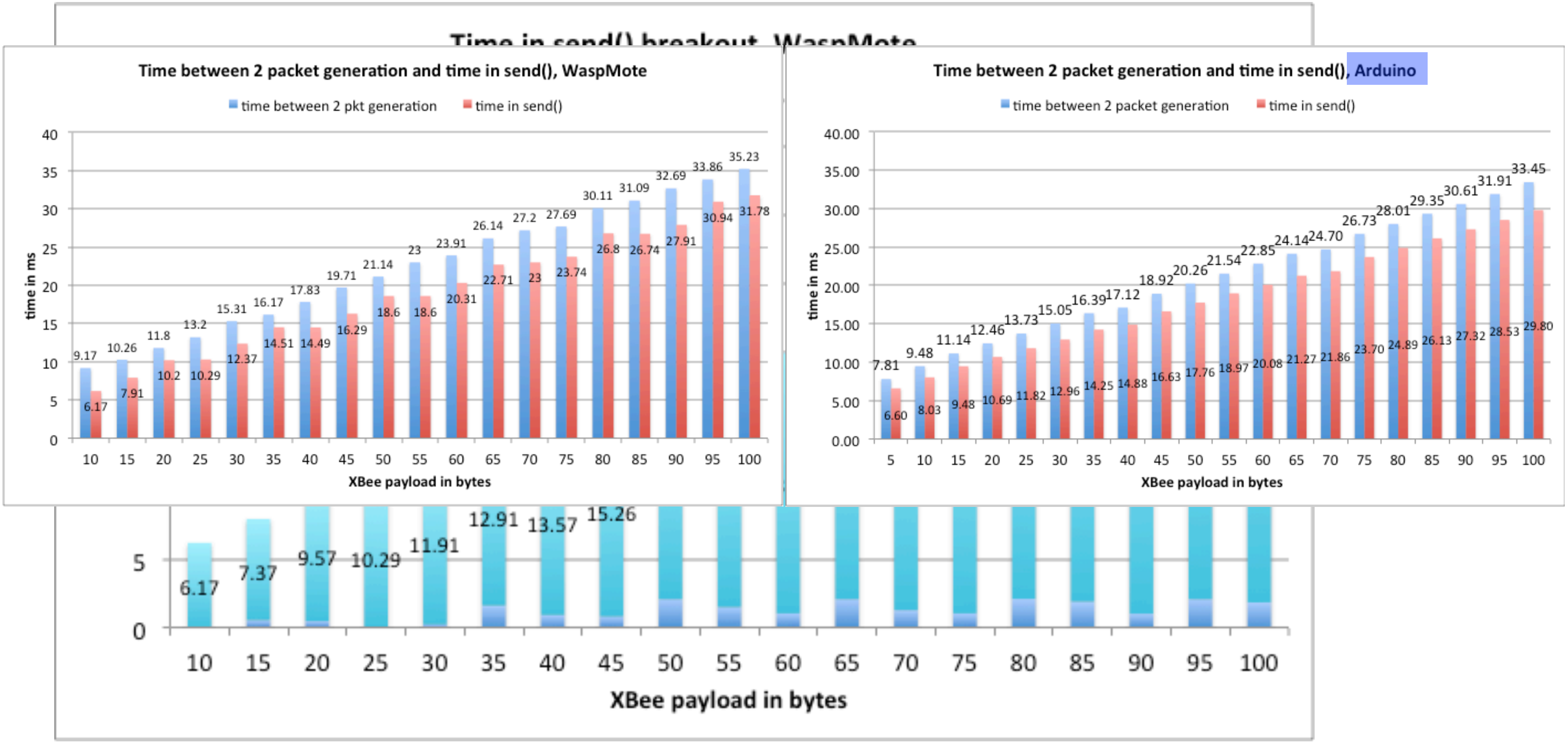
Sending performance



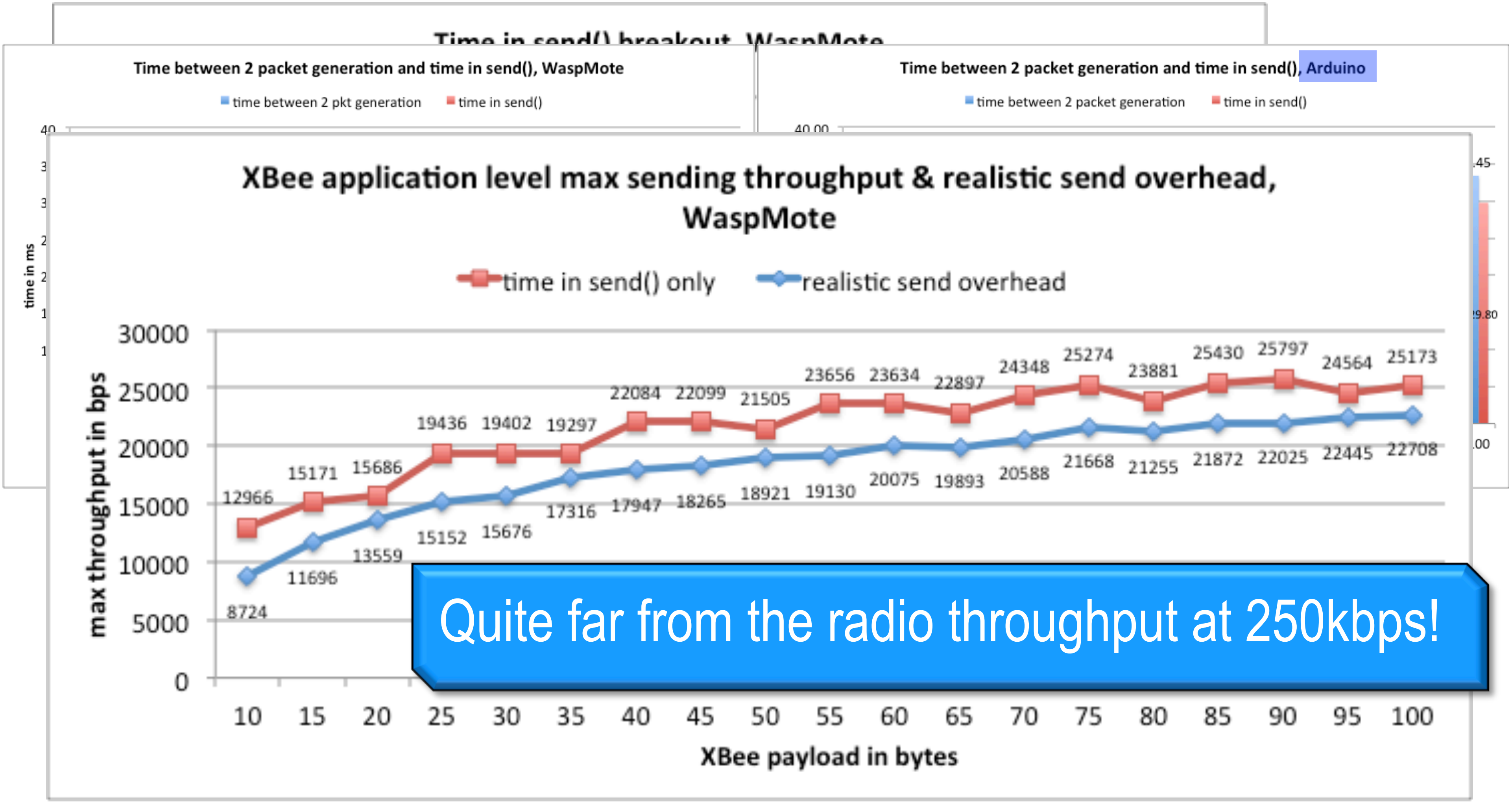
Sending performance



Sending performance



Sending performance



Quite far from the radio throughput at 250kbps!

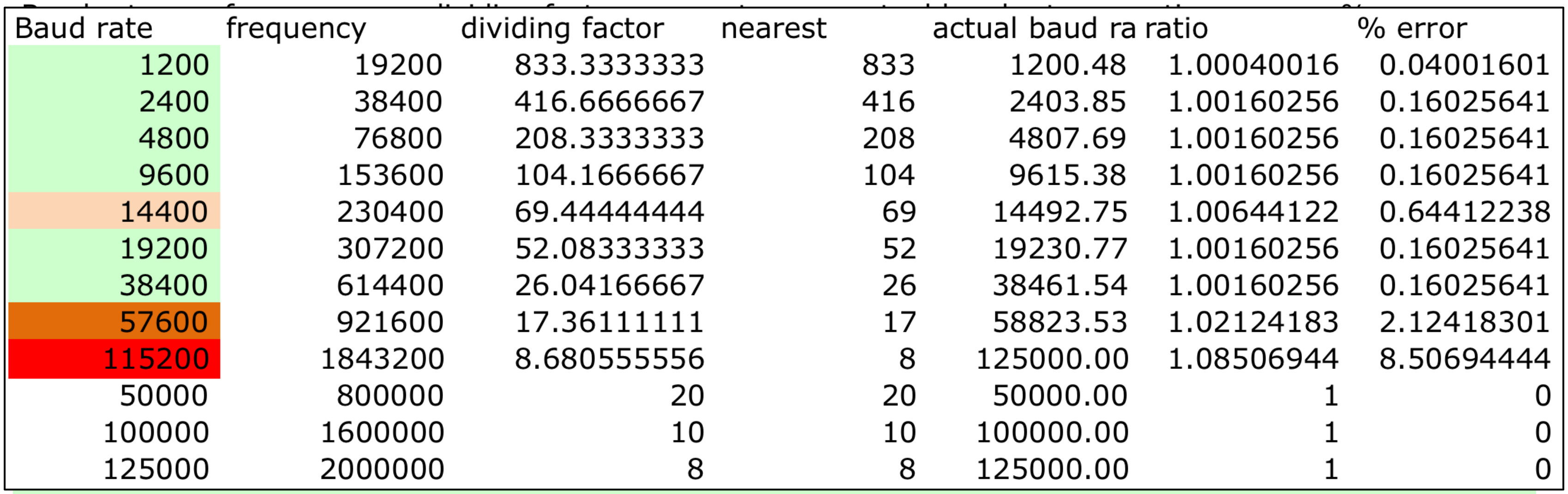
Improving sending performances

- Xbee modules require the frequency to be 16 times the baud rate: 38400→614400Hz
- WaspMote are 8MHz and Arduino are 16Mhz

Baud rate	frequency	dividing factor	nearest	actual baud rate	ratio	% error
1200	19200	416.666667	416	1201.92	1.00160256	0.16025641
2400	38400	208.333333	208	2403.85	1.00160256	0.16025641
4800	76800	104.166667	104	4807.69	1.00160256	0.16025641
9600	153600	52.0833333	52	9615.38	1.00160256	0.16025641
14400	230400	34.7222222	34	14705.88	1.02124183	2.12418301
19200	307200	26.0416667	26	19230.77	1.00160256	0.16025641
38400	614400	13.0208333	13	38461.54	1.00160256	0.16025641
57600	921600	8.68055556	8	62500.00	1.08506944	8.50694444
115200	1843200	4.34027778	4	125000.00	1.08506944	8.50694444
100000	1600000	5	5	100000.00	1	0
125000	2000000	4	4	125000.00	1	0
250000	4000000	2	2	250000.00	1	0

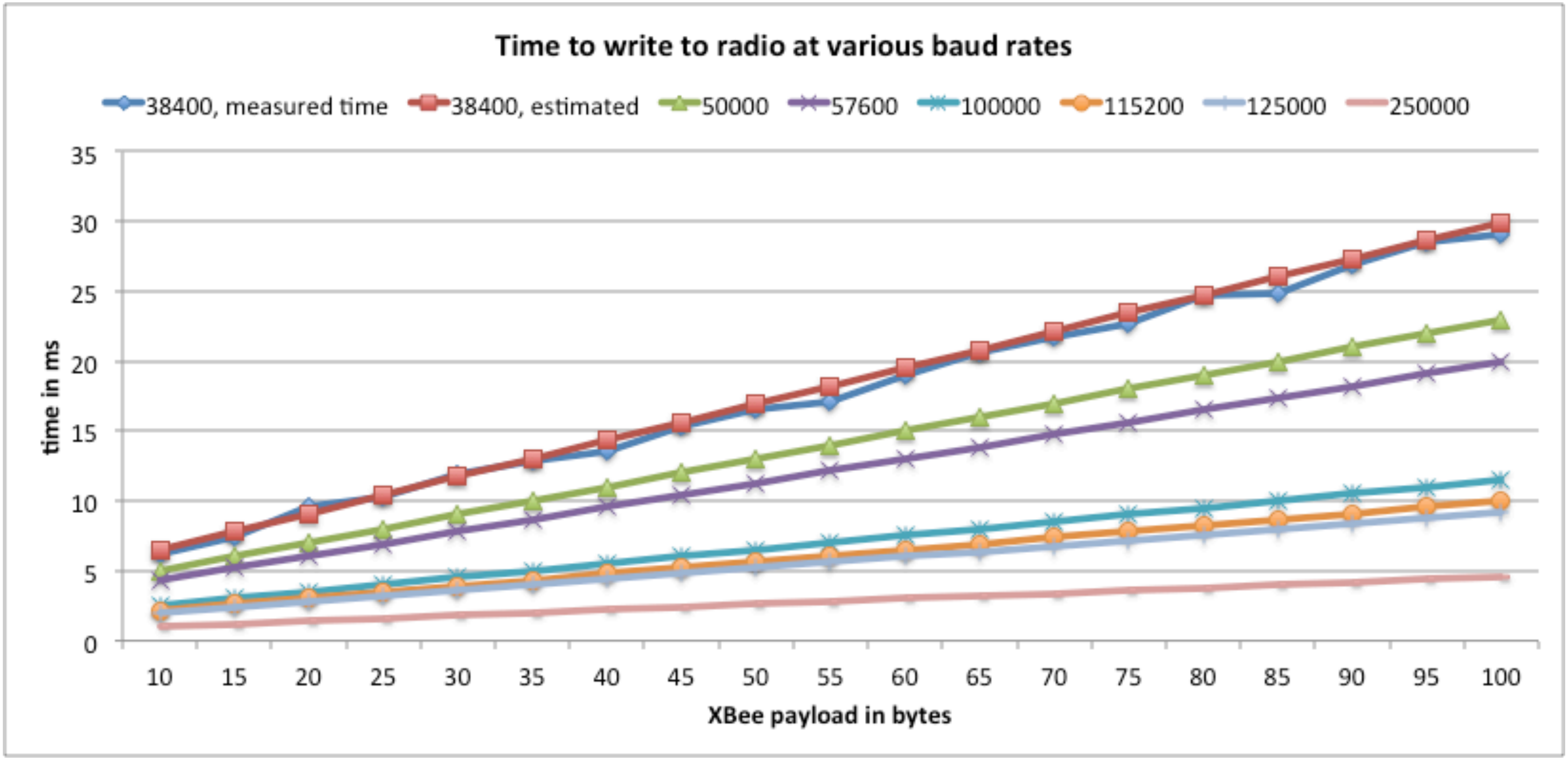
Improving sending performances

- □ Xbee modules require the frequency to be 16 times the baud rate: 38400→614400Hz
- □ WaspMote are 8MHz and Arduino are 16Mhz



Baud rate	frequency	dividing factor	nearest	actual baud ra	ratio	% error
1200	19200	833.3333333	833	1200.48	1.00040016	0.04001601
2400	38400	416.6666667	416	2403.85	1.00160256	0.16025641
4800	76800	208.3333333	208	4807.69	1.00160256	0.16025641
9600	153600	104.1666667	104	9615.38	1.00160256	0.16025641
14400	230400	69.44444444	69	14492.75	1.00644122	0.64412238
19200	307200	52.08333333	52	19230.77	1.00160256	0.16025641
38400	614400	26.04166667	26	38461.54	1.00160256	0.16025641
57600	921600	17.36111111	17	58823.53	1.02124183	2.12418301
115200	1843200	8.680555556	8	125000.00	1.08506944	8.50694444
50000	800000	20	20	50000.00	1	0
100000	1600000	10	10	100000.00	1	0
125000	2000000	8	8	125000.00	1	0

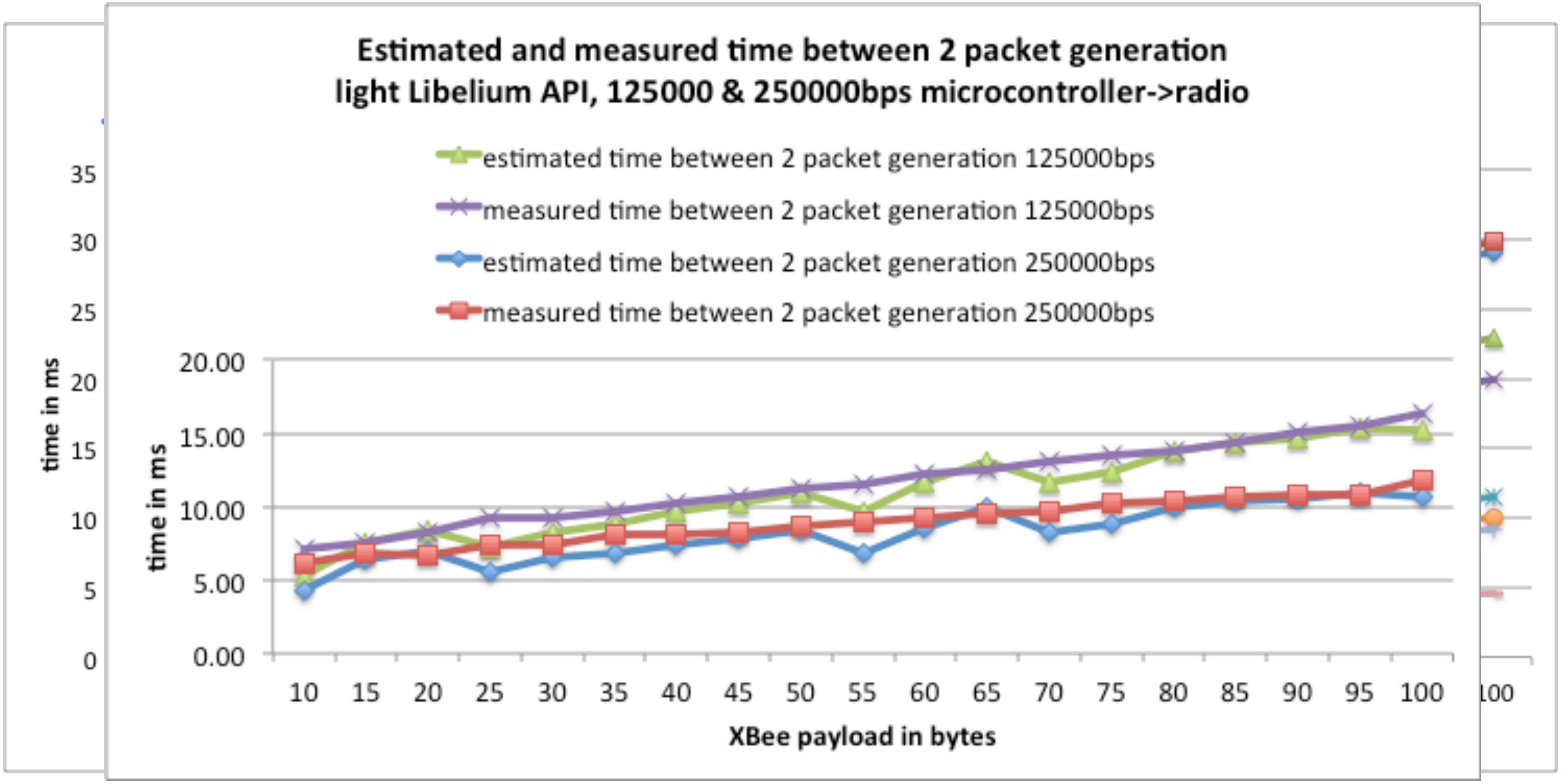
Sending performances at maximum UART speed



$$t_{send}^{B} = t_{send}^{38400} - timeToWriteToRadio^{38400} + timeToWriteToRadio^{B}$$

$$t_{pkt}^{B} = t_{pkt}^{38400} - t_{send}^{38400} + t_{send}^{B}$$

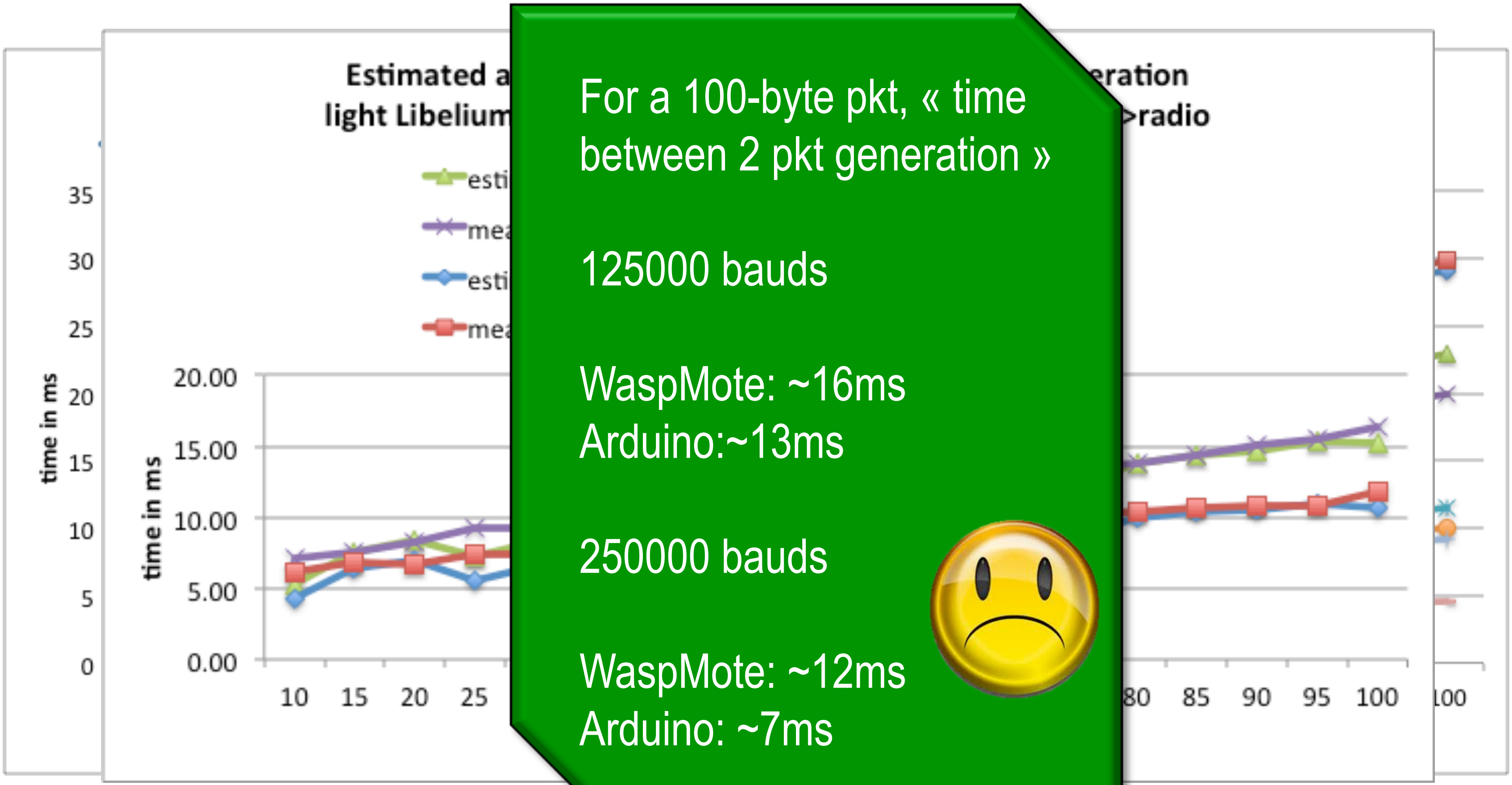
Sending performances at maximum UART speed



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

$$t^{B}_{pkt} = \quad t^{38400}_{pkt} - t^{38400}_{send} + t^{B}_{send}$$

Sending performances at maximum UART speed



For a 100-byte pkt, « time between 2 pkt generation »

125000 bauds

WaspMote: ~16ms
Arduino:~13ms

250000 bauds

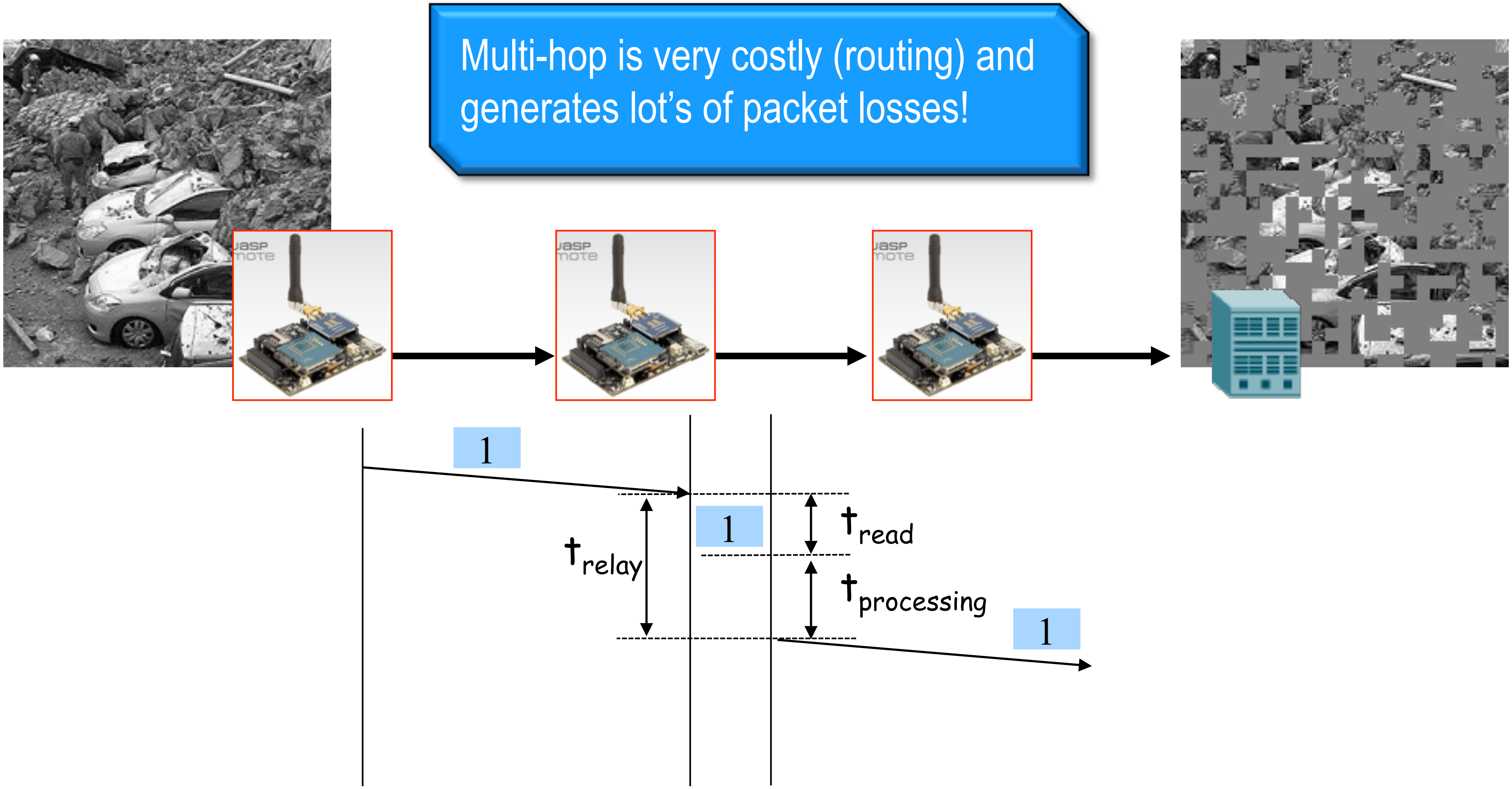
WaspMote: ~12ms
Arduino: ~7ms

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

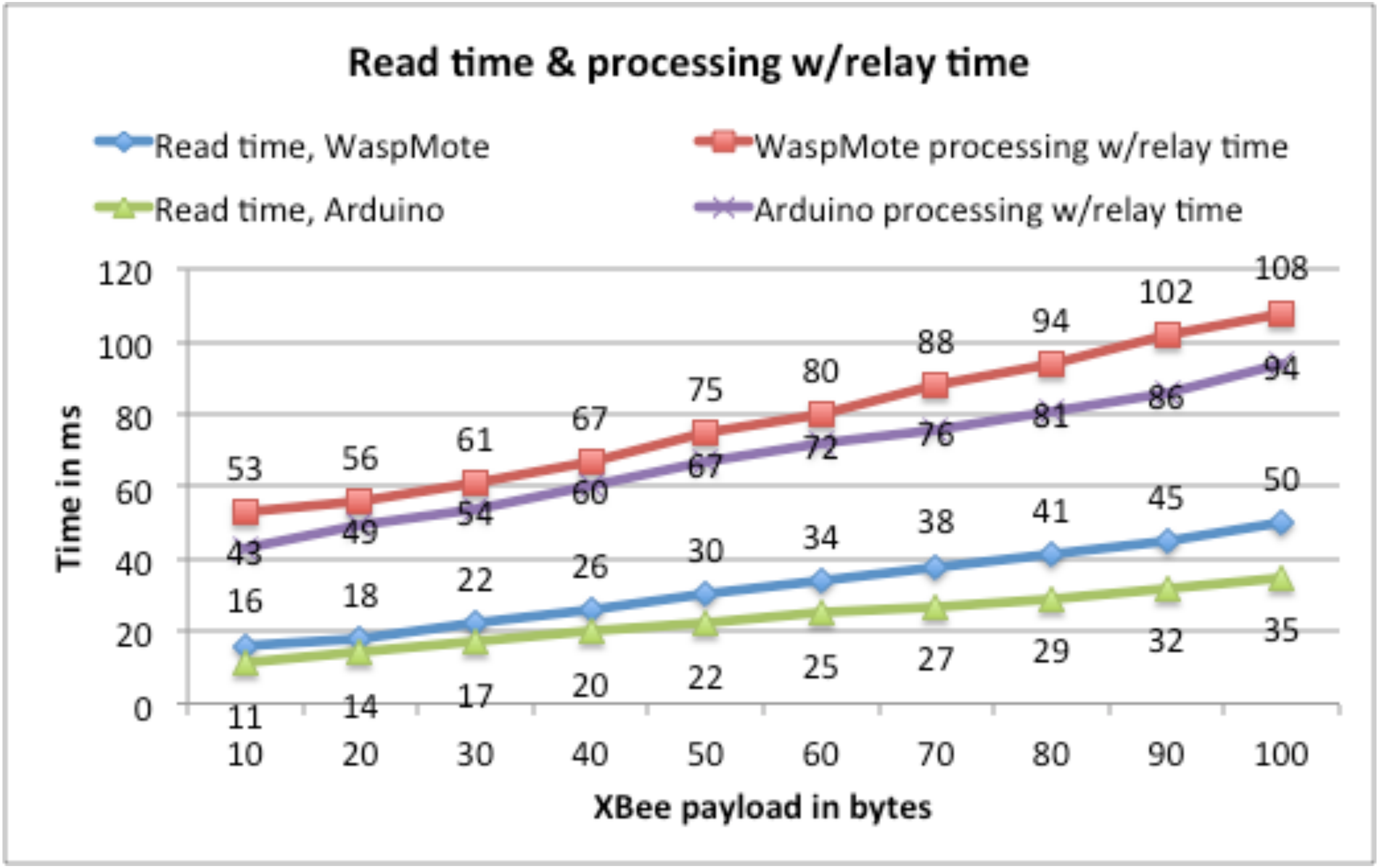
$$t^{B}_{pkt} = t^{38400}_{pkt} - t^{38400}_{send} + t^{B}_{send}$$

Multi-Hop Packet Forwarding

Multi-hop is very costly (routing) and generates lot's of packet losses!

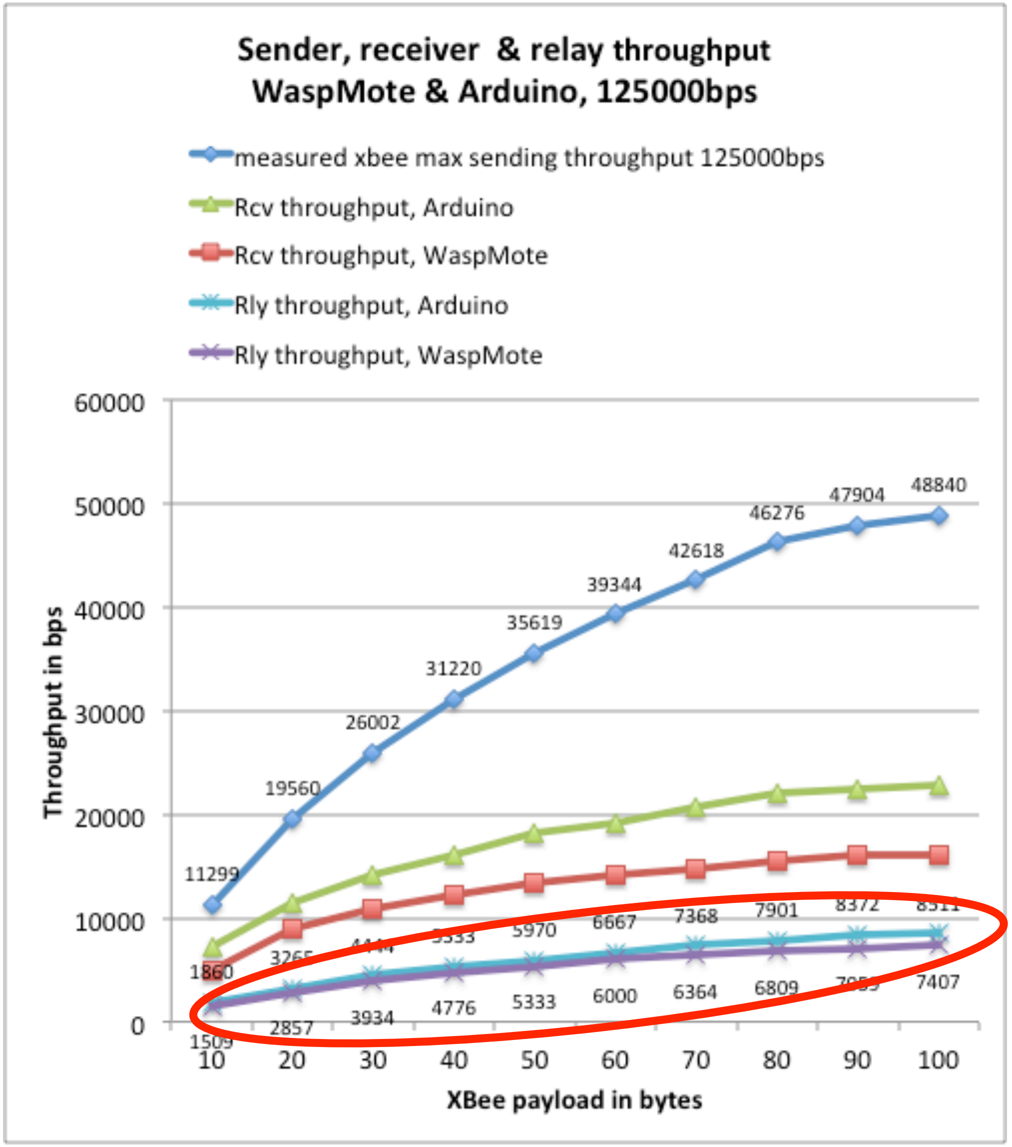


Read time and relay time



Read time is quite independant from the UART baud rate

Maximum expected throughput



Limitations on image transmission

Original BMP 16384b

128x128
16384b
Between 14s and 16s!

Relay latency is about 90ms-110ms per packet!

Original BMP 40000b

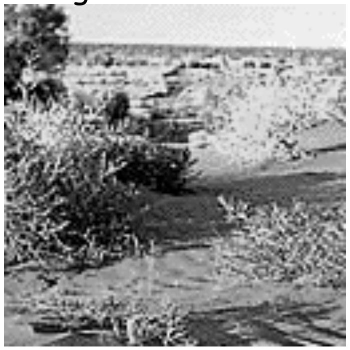
200x200
40000b
Between 36s and 44s !

Robust image encoding

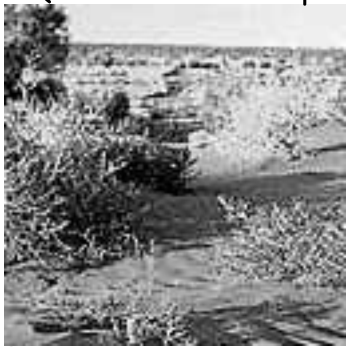
- JPEG-like coder, but on 8x8 pixel blocks
- Arai-Agui-Nakajima DCT is used with fixed-point arithmetic
- binary encoding operations is reduced by using jointly Golomb and Multiple Quantization
- block interleaving method
- A Quality Factor can be used to tuned the whole process

Dynamic Quality Factor 128x128 - 90B payload

Original BMP 16384b

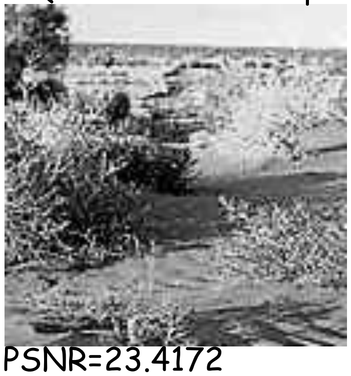


Q=50 S=4800b 63pkts



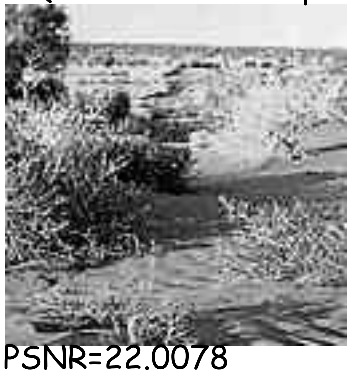
PSNR=24.6765

Q=40 S=4268b 56pkts



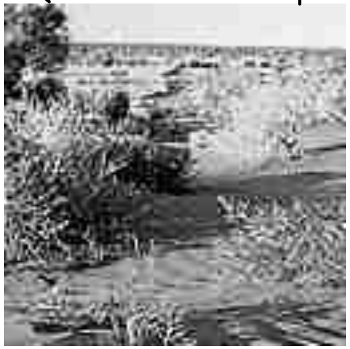
PSNR=23.4172

Q=30 S=3604b 46pkts



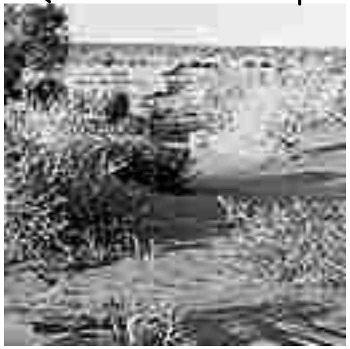
PSNR=22.0078

Q=20 S=2781b 34pkts



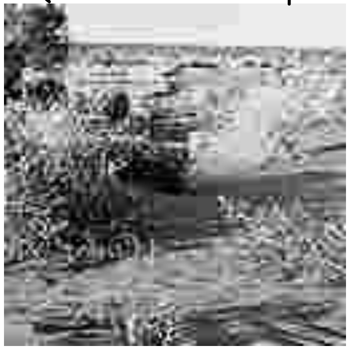
PSNR=20.4087

Q=15 S=2268b 28pkts



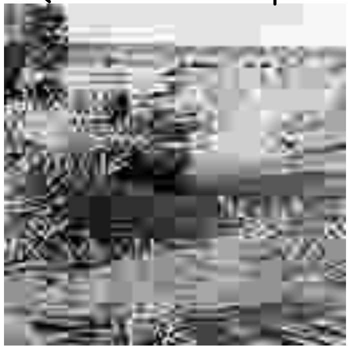
PSNR=19.5864

Q=10 S=1757b 12pkts



PSNR=18.6861

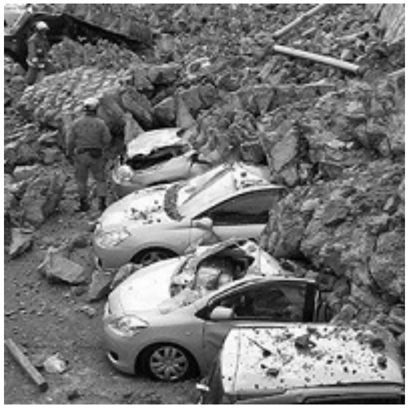
Q=5 S=1006b 12pkts



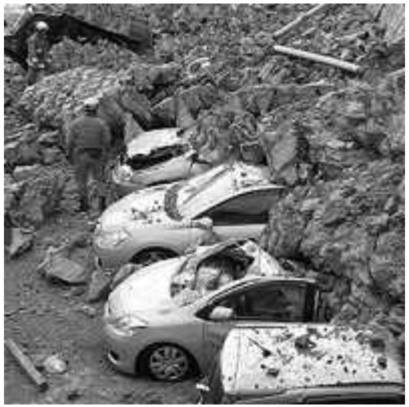
PSNR=17.3283

Dynamic Quality Factor 200x200 - 90B payload

Original BMP 40000b

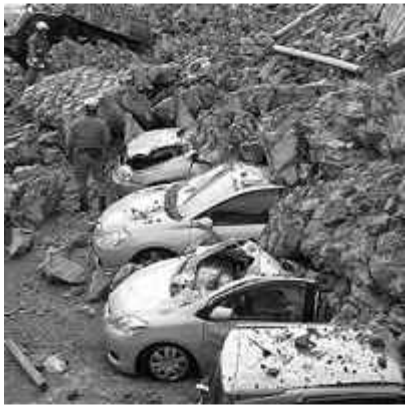


Q=50 S=11045b 142pkts



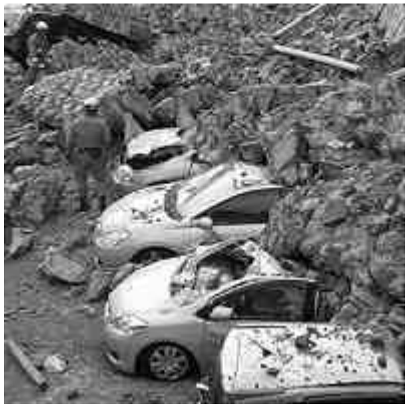
PSNR=25.1661

Q=40 S=9701b 123pkts



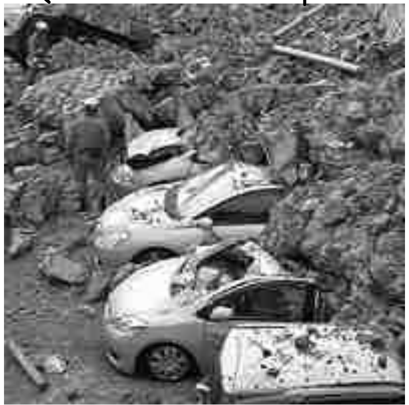
PSNR=24.2231

Q=30 S=8100b 101pkts



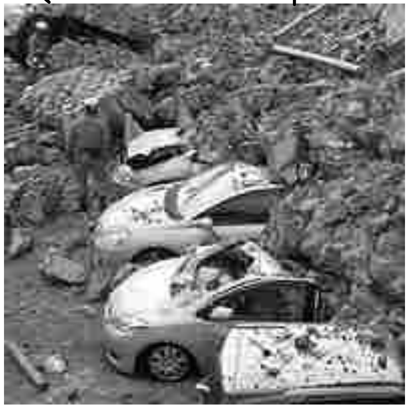
PSNR=23.2264

Q=20 S=6236b 76pkts



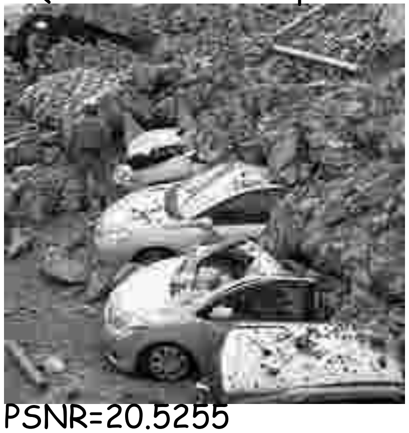
PSNR=22.1293

Q=15 S=5188b 63pkts



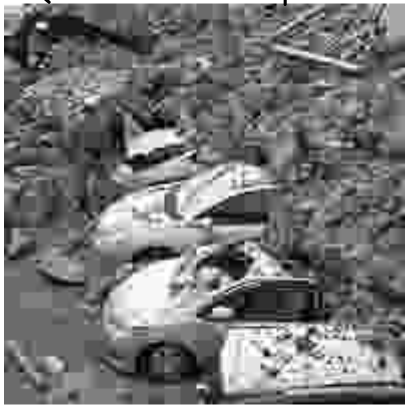
PSNR=21.4475

Q=10 S=3868b 47pkts



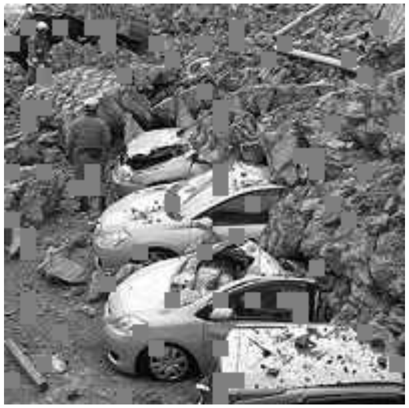
PSNR=20.5255

Q=5 S=2053b 24pkts

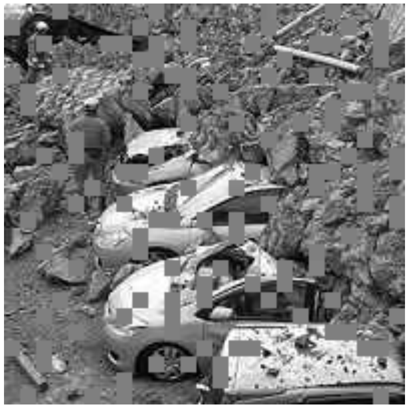


PSNR=18.937

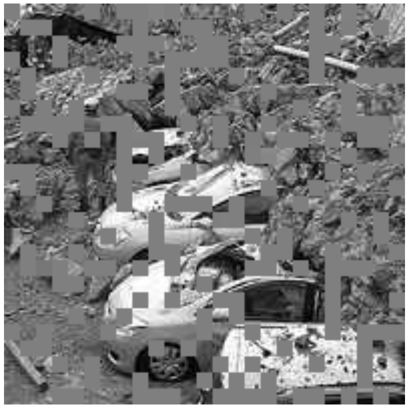
Impact of pkt losses



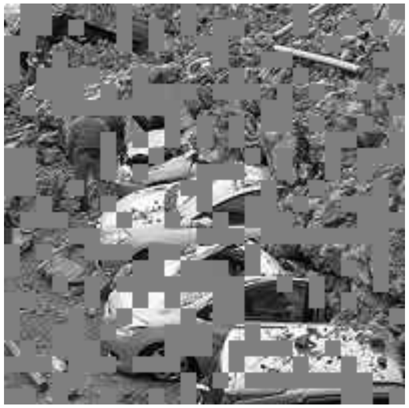
10%



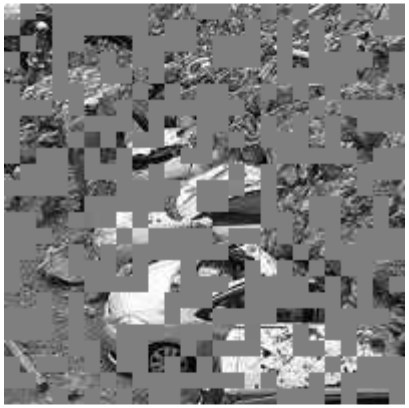
20%



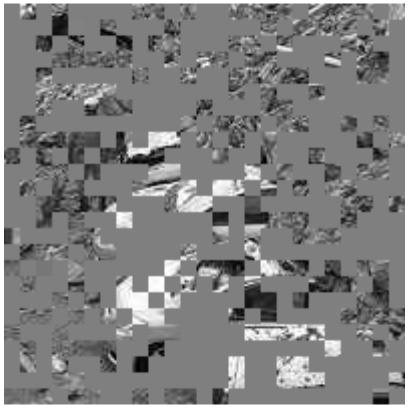
30%



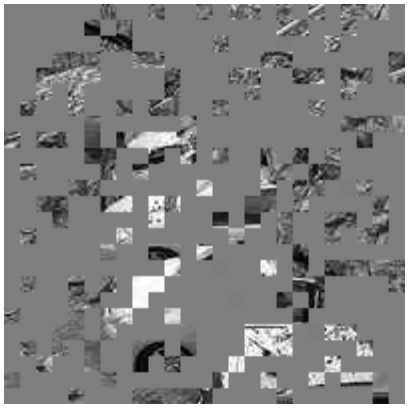
40%



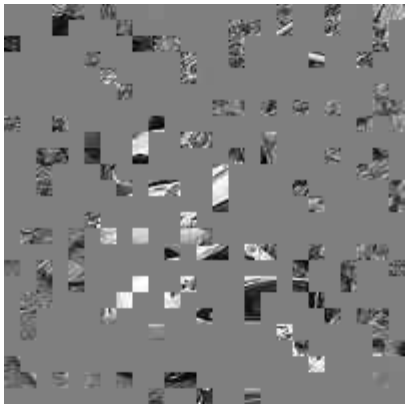
50%



60%



70%

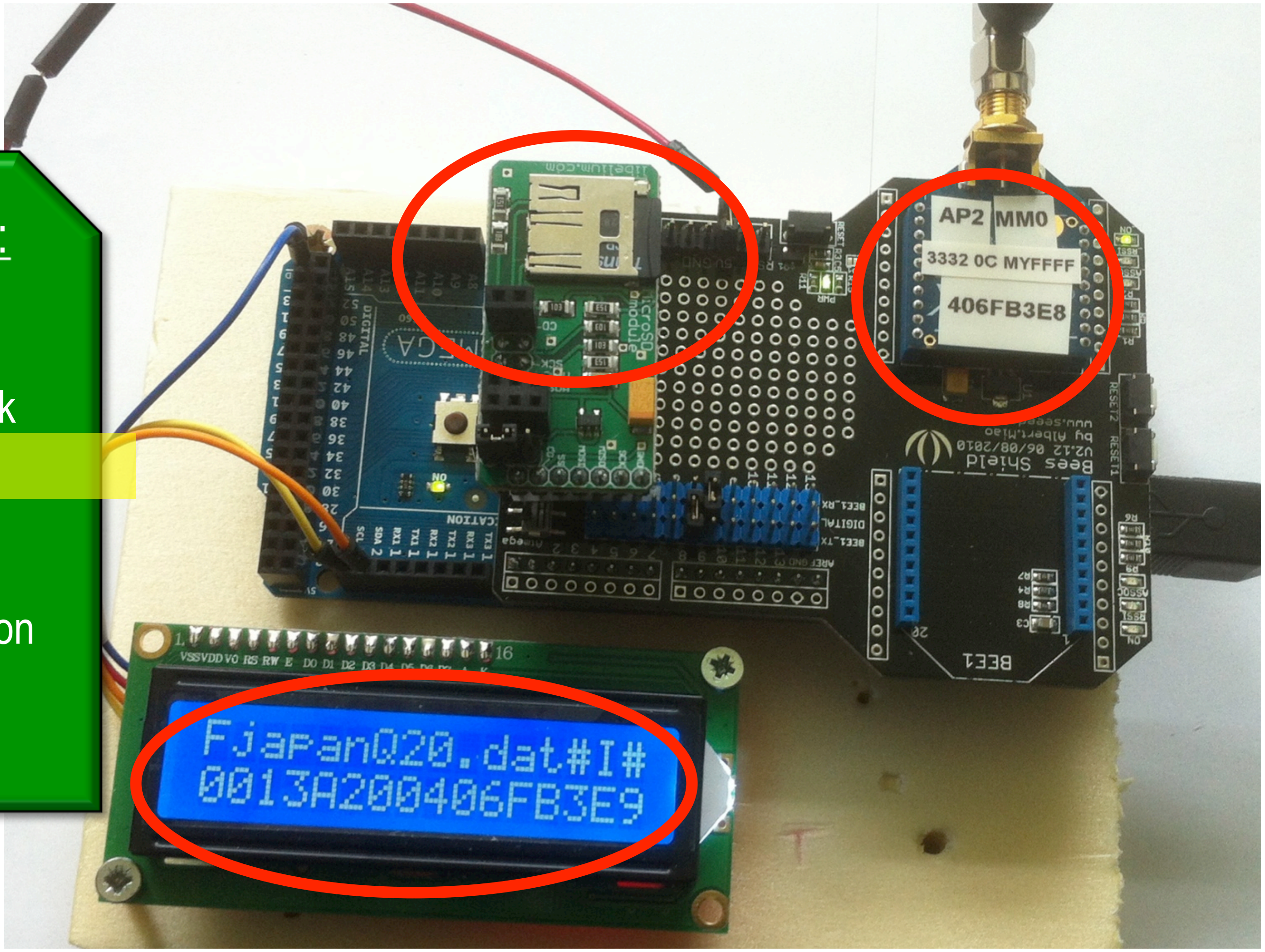


80%

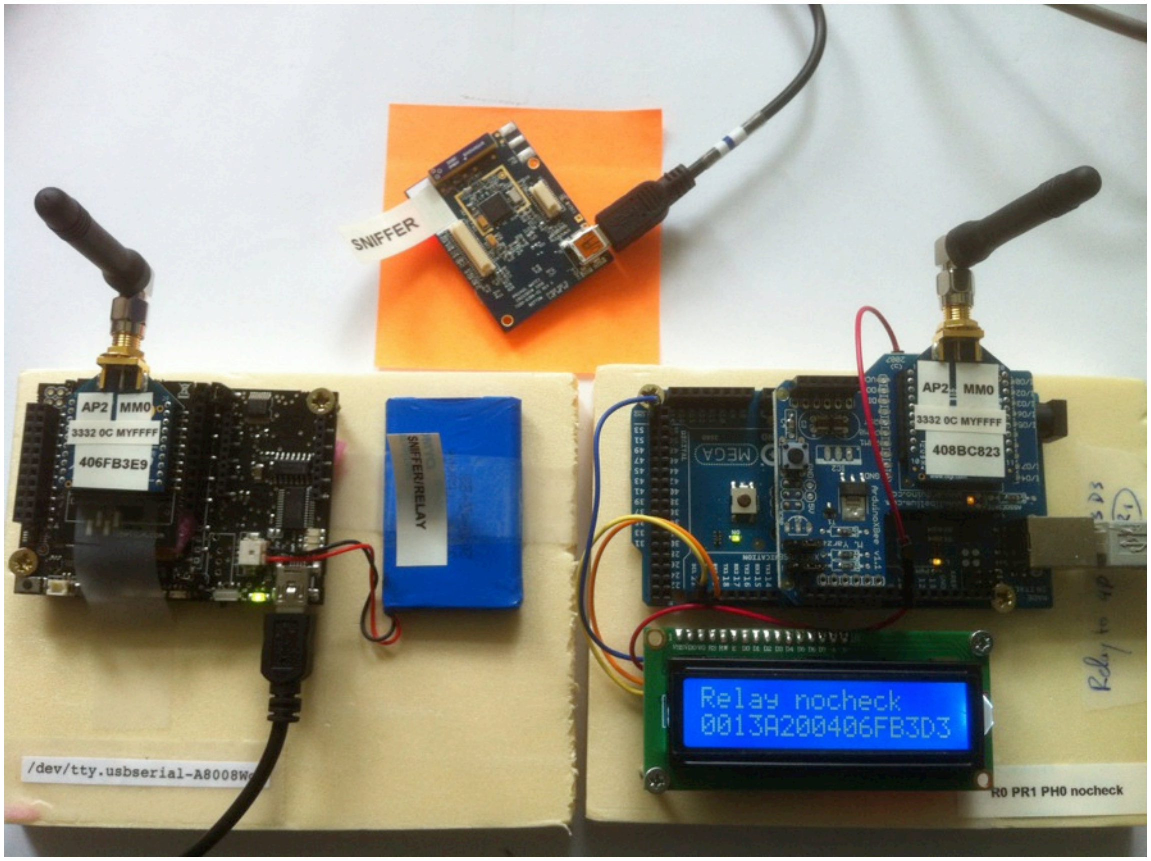
FILE SENDER NODE

<u>Fully configurable:</u>

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



Relay nodes

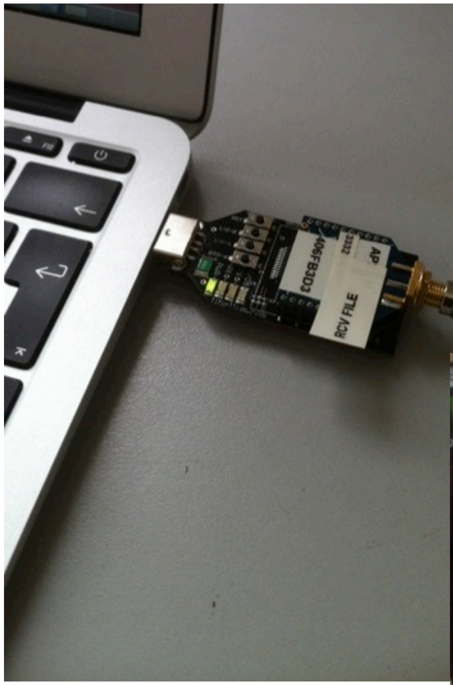


<u>Fully configurable:</u>

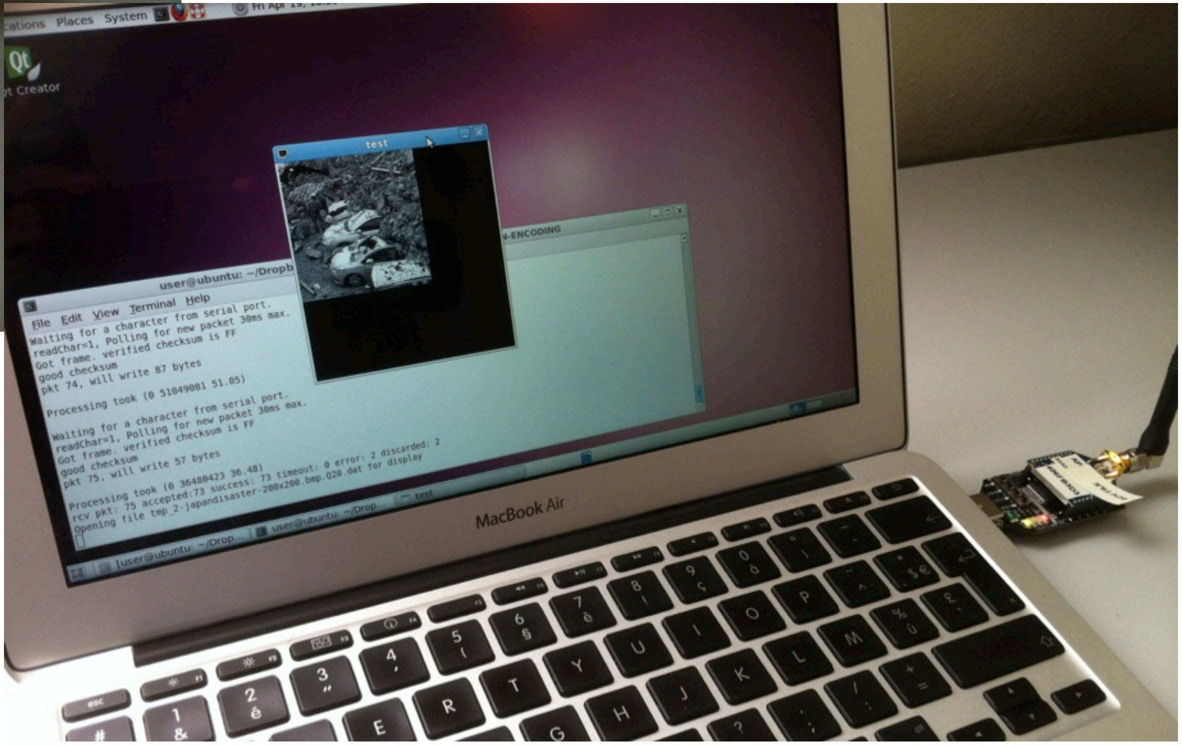
Destination node
Additional relay delay
Clock synchronization

Libelium WaspMote, Arduino, Imote2

Sink node

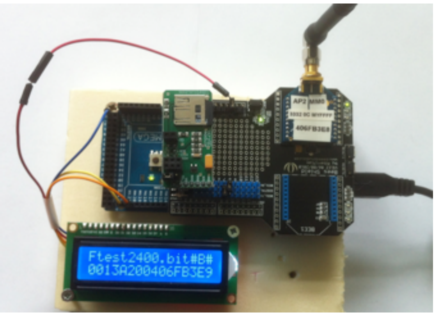


Linux PC/Laptop with USB/Serial gateway

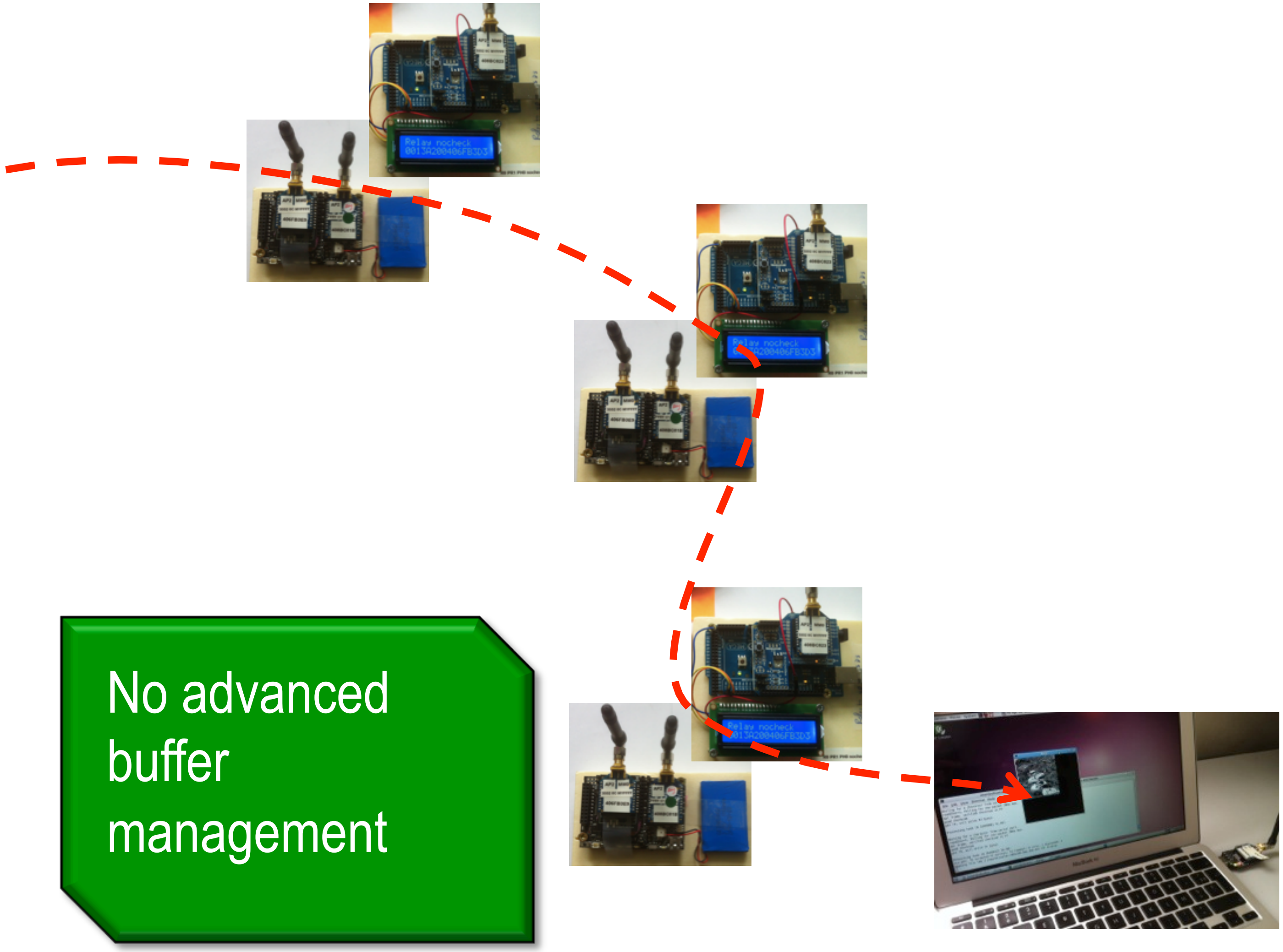


```
Waiting for a character from serial port.
readChar=1, Polling for new packet 30ms max.
Got frame. verified checksum is FF
good checksum
pkt 74, will write 87 bytes

Processing took (0 51049081 51.05)

Waiting for a character from serial port.
readChar=1, Polling for new packet 30ms max.
Got frame. verified checksum is FF
good checksum
pkt 75, will write 57 bytes

Processing took (0 36480423 36.48)
rcv pkt: 75 accepted:73 success: 73 timeout: 0 error: 2 discarded: 2
Opening file tmp_2-japandisaster-200x200.bmp.020.dat for display
```

Test-bed



Q=20 S=6236b 76pkts



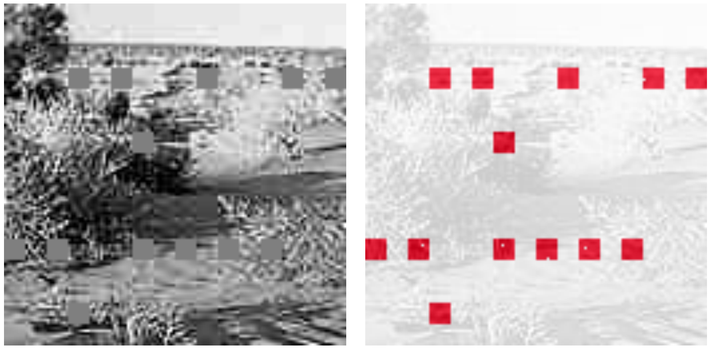
No advanced buffer management

Motivations

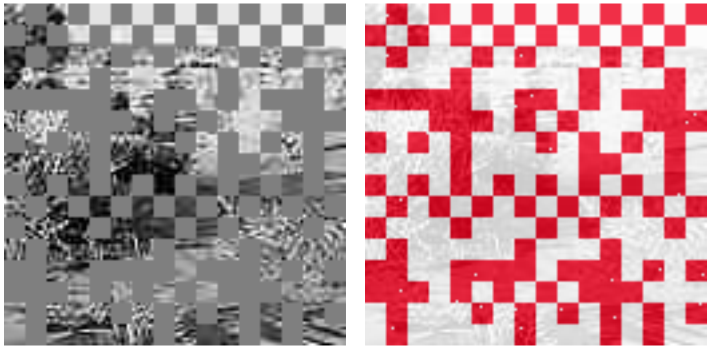
- ❑ Need a controlled environment
  - ❑ Test multi-source scenario
  - ❑ Quantify impact of radio interference
  - ❑ Test multi-path routing
  - ❑ Determine typical latencies
- ❑ Adopt a « fully controllable » approach
  - ❑ Each node can be dynamically configured…
  - ❑ … to « know » what is going on.

Experimental results Q=20, 34 pkts

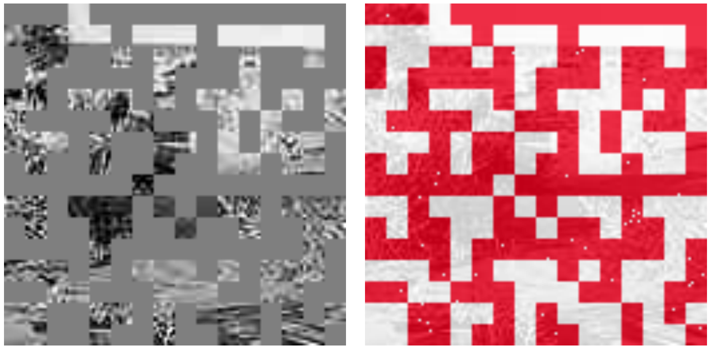
WaspMote relay node. Relay time T_R is 102ms-111ms



110ms PSNR=25.2272



100ms PSNR=15.4364

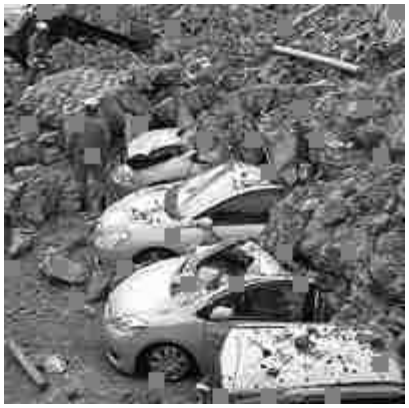


90ms
PSNR=14.1088

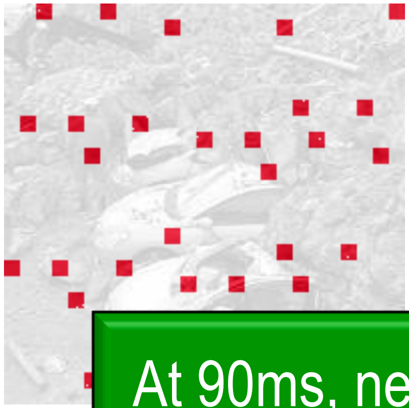
At 110ms, need 3.86s to send the image. 1-hop latency is $3.86+T_R$

Experimental results
Q=20, 76 pkts

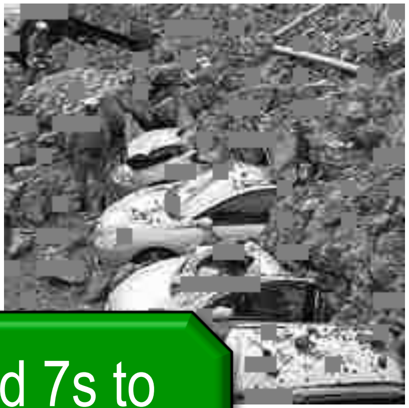
Arduino relay node. Relay time T_R is 92ms-100ms



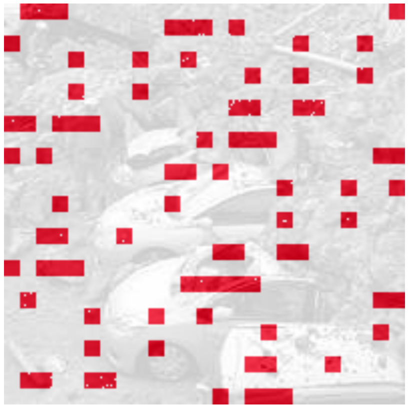
90ms



PS

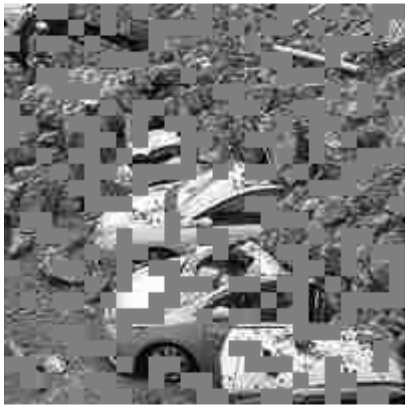


80ms

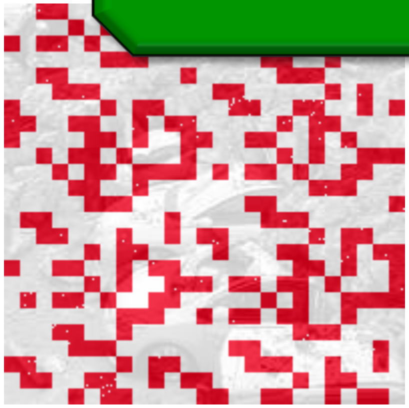


PSNR=21.9901

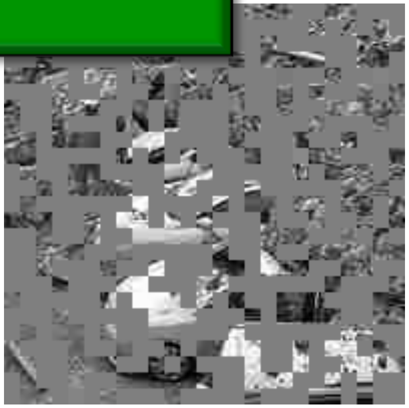
At 90ms, need 7s to send the image.



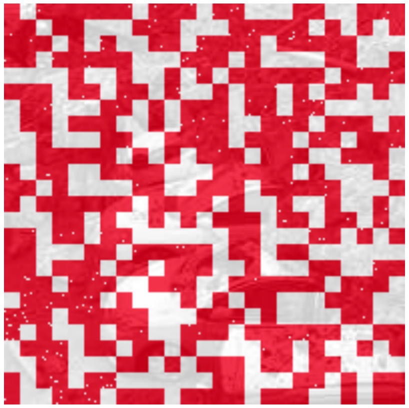
70ms



PSNR=17.265



60ms



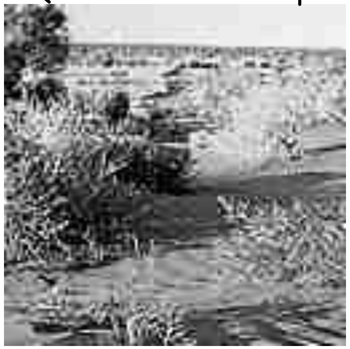
PSNR=14.2429

Intrusion detection applications

- 128x128 image with Arduino relay node can be sent in about 3.18s if inter-pkt time is 90ms
- With inter-pkt time of 80ms and a Quality Factor of 10, the image can be sent in about 1s!

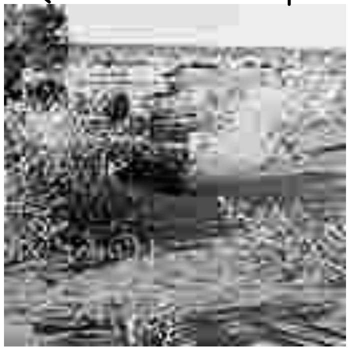
Original BMP 16384b

Q=20 S=2781b 34pkts



PSNR=20.4087

Q=10 S=1757b 12pkts

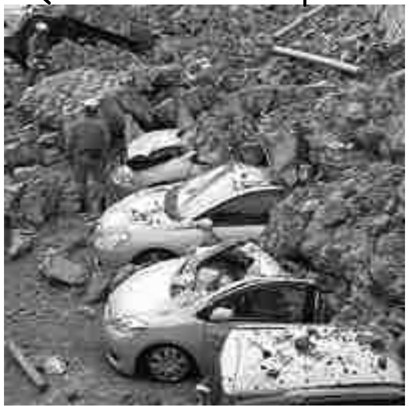


PSNR=18.6861

Situation awareness applications

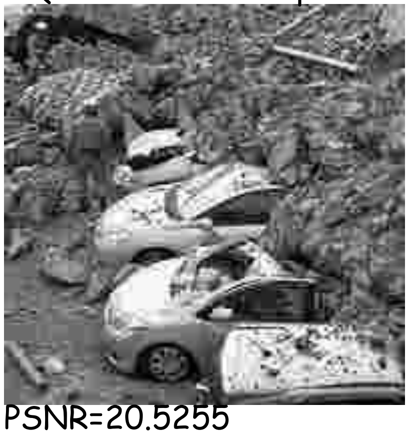
- With Q=20 and inter-pkt time of 90ms, an 200x200 image can be sent in 7s
- Need about 12min to get images from 100 different locations if appropriate scheduling is used
- Again, can decrease quality factor (Q=10) or inter-pkt time to 80ms (7min) or even 70ms (5.5min)

Q=20 S=6236b 76pkts



PSNR=22.1293

Q=10 S=3868b 47pkts



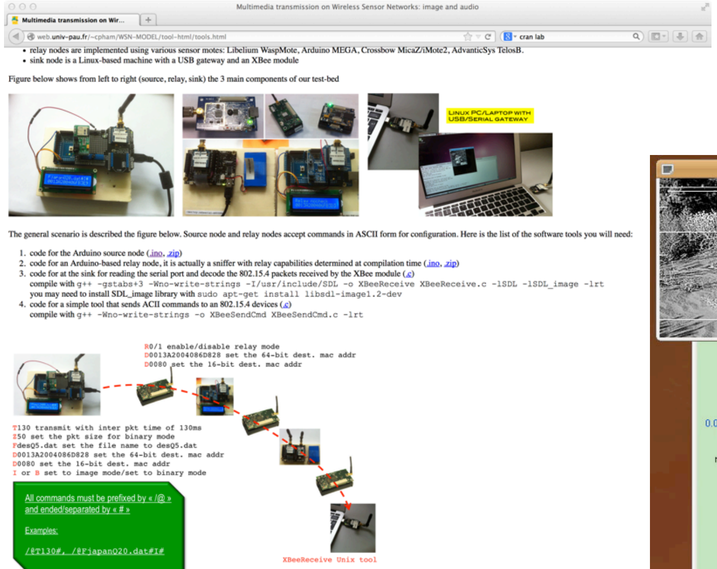
PSNR=20.5255

Conclusions

- Low-cost wireless sensor nodes have limited communication performances
- Important to determine the maximum performance level one can get at the application level
- Traditional congestion control methods may be not adequate
- We are investigating mutual-exclusion or smart selection approaches to avoid simultaneous image sending within the same area

Some links

http://web.univ-pau.fr/~cpham/WSN-MODEL/tool-html/tools.html


Multimedia transmission on Wireless Sensor Networks: image and audio
Multimedia transmission on Wir...
web.univ-pau.fr/~cpham/WSN-MODEL/tool-html/tools.html
cran lab
LINUX PC/LAPTOP WITH USB/SERIAL GATEWAY

- relay nodes are implemented using various sensor motes: Libelium WaspMote, Arduino MEGA, Crossbow MicaZ/iMote2, AdvanticSys TelosB.
- sink node is a Linux-based machine with a USB gateway and an XBee module

Figure below shows from left to right (source, relay, sink) the 3 main components of our test-bed

The general scenario is described the figure below. Source node and relay nodes accept commands in ASCII form for configuration. Here is the list of the software tools you will need:

1. code for the Arduino source node (.ino, .zip)
2. code for an Arduino-based relay node, it is actually a sniffer with relay capabilities determined at compilation time (.ino, .zip)
3. code for at the sink for reading the serial port and decode the 802.15.4 packets received by the XBee module (.c)
   compile with `g++ -gstabs+3 -Wno-write-strings -I/usr/include/SDL -o XBeeReceive XBeeReceive.c -lSDL -lSDL_image -lrt`
   you may need to install SDL_image library with `sudo apt-get install libsdl-image1.2-dev`
4. code for a simple tool that sends ACII commands to an 802.15.4 devices (.c)
   compile with `g++ -Wno-write-strings -o XBeeSendCmd XBeeSendCmd.c -lrt`

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

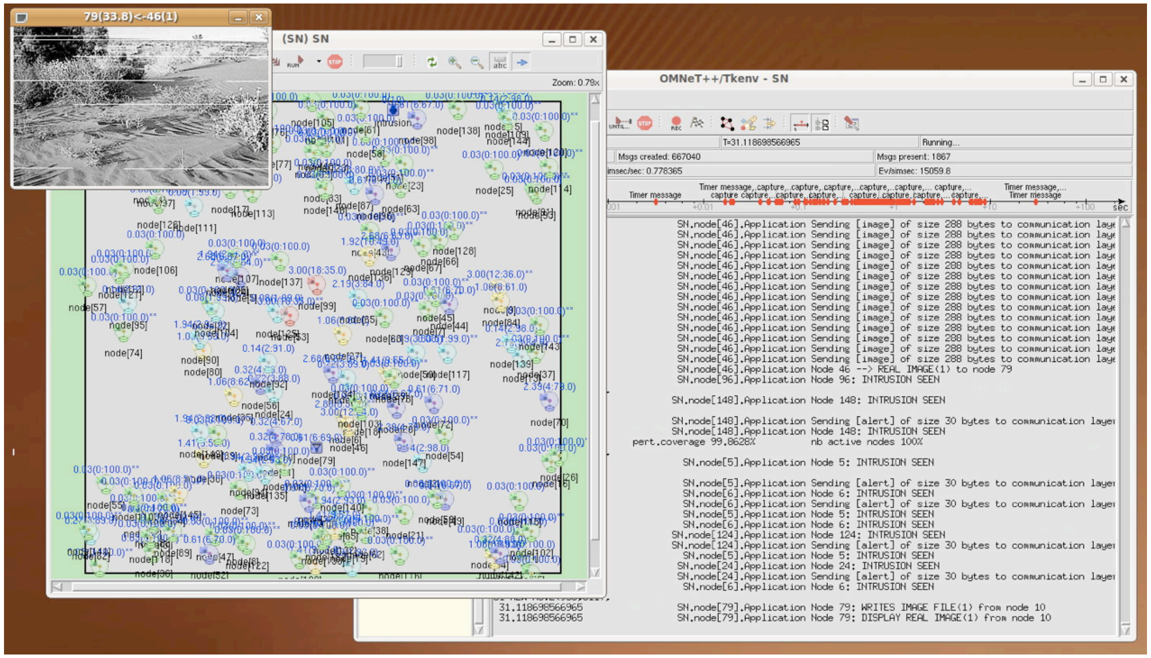
```
T130 transmit with inter pkt time of 130ms
Z50 set the pkt size for binary mode
FdesQ5.dat set the file name to desQ5.dat
D0013A2004086D828 set the 64-bit dest. mac addr
D0080 set the 16-bit dest. mac addr
I or B set to image mode/set to binary mode
```

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

Examples:

/@T130#, /@FjapanQ20.dat#I#

XBeeReceive Unix tool


79(33.8)<-46(1)
(SN) SN
Zoom: 0.79x
OMNeT++/Tkenv - SN
UNTIL...
STOP
REC
Timer message
Timer message,..capture,..capture,.capture,..capture,..capture,... capture,...
capture capture,..capture,.capture,... capture,.capture,.capture,... capture,...
Timer message,...
Timer message
+0.01
+0.1
+1
+10
+100
sec

```
T=31.118698566965          Running...
Msgs created: 667040       Msgs present: 1867
imsec/sec: 0.778365        Ev/simsec: 15059.8
```

```
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Sending [image] of size 288 bytes to communication laye
                SN.node[46].Application Node 46 --> REAL IMAGE(1) to node 79
                SN.node[96].Application Node 96: INTRUSION SEEN

               SN.node[148].Application Node 148: INTRUSION SEEN

               SN.node[148].Application Sending [alert] of size 30 bytes to communication layer
               SN.node[148].Application Node 148: INTRUSION SEEN
      pert.coverage 99.8628%        nb active nodes 100%

                 SN.node[5].Application Node 5: INTRUSION SEEN

                 SN.node[5].Application Sending [alert] of size 30 bytes to communication layer
                 SN.node[6].Application Node 6: INTRUSION SEEN
                 SN.node[6].Application Sending [alert] of size 30 bytes to communication layer
                 SN.node[5].Application Node 5: INTRUSION SEEN
                 SN.node[6].Application Node 6: INTRUSION SEEN
               SN.node[124].Application Node 124: INTRUSION SEEN
               SN.node[124].Application Sending [alert] of size 30 bytes to communication layer
                 SN.node[5].Application Node 5: INTRUSION SEEN
                SN.node[24].Application Node 24: INTRUSION SEEN
                SN.node[24].Application Sending [alert] of size 30 bytes to communication layer
                 SN.node[6].Application Node 6: INTRUSION SEEN

31.118698566965 SN.node[79].Application Node 79: WRITES IMAGE FILE(1) from node 10
31.118698566965 SN.node[79].Application Node 79: DISPLAY REAL IMAGE(1) from node 10
```

http://web.univ-pau.fr/~cpham/WSN-MODEL/wvsn-castalia.html