

COMMUNICATION PERFORMANCE OF LOW-RESOURCE SENSOR MOTES FOR DATA-INTENSIVE APPLICATIONS

C. PHAM

IFIP/IEEE WIRELESS DAYS 2013

NOVEMBER 15TH, 2013

VALENCIA, SPAIN



PROF. CONGDUC PHAM
[HTTP://WWW.UNIV-PAU.FR/~CPHAM](http://www.univ-pau.fr/~cpham)
UNIVERSITÉ DE PAU, FRANCE



SEARCH & RESCUE, SITUATION AWARENESS



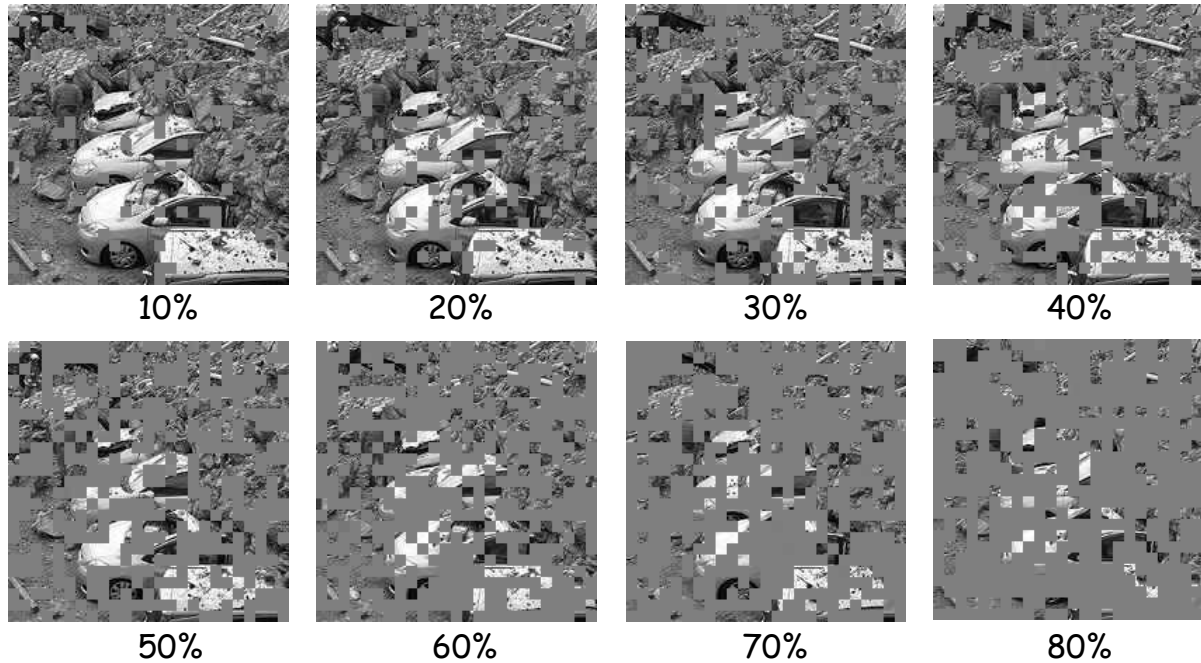
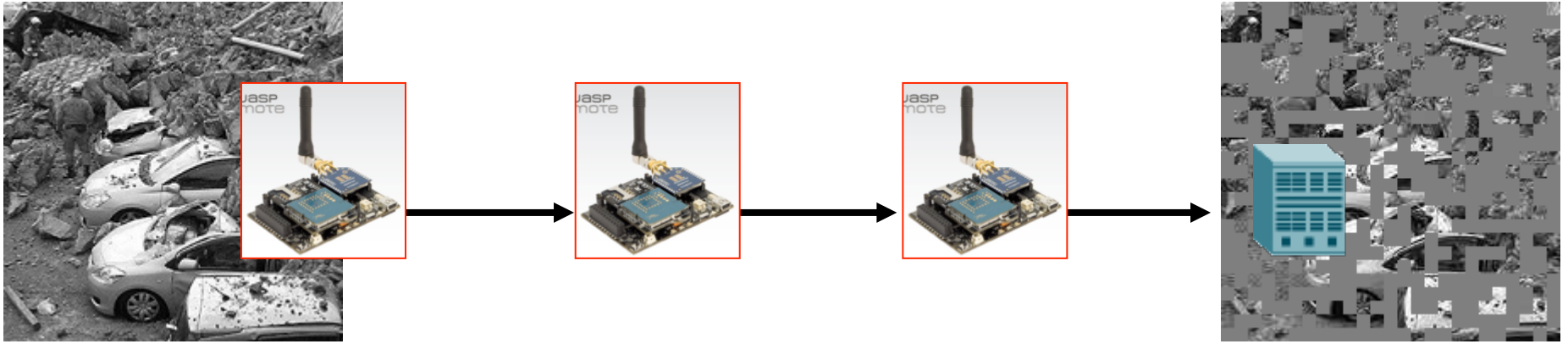
Imote2



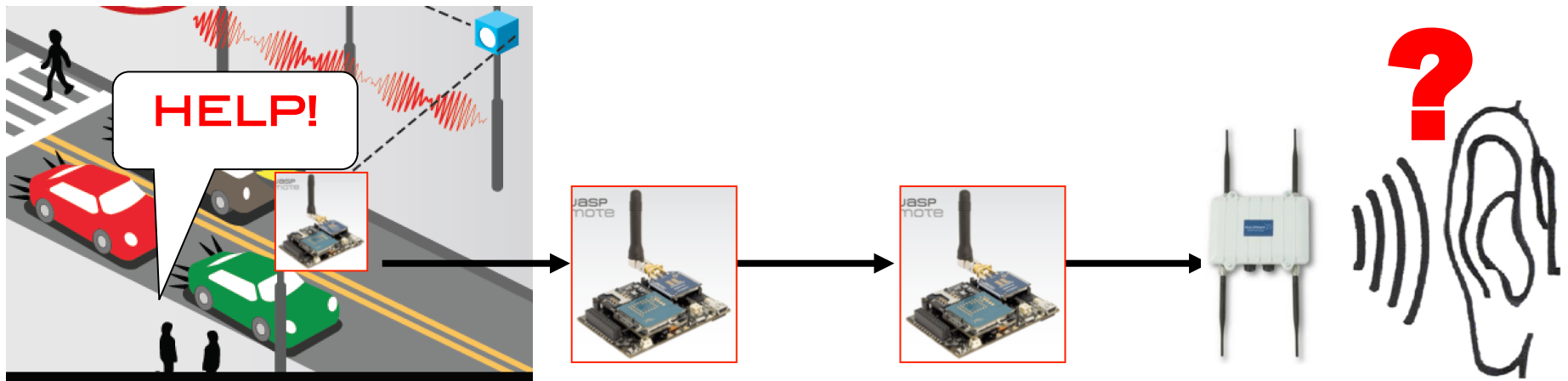
Multimedia
board



IMAGE QUALITY, DELIVERY LATENCIES,...



EAR-IT: AUDIO SURVEILLANCE IN SMARTCITIES AND SMARTBUILDINGS



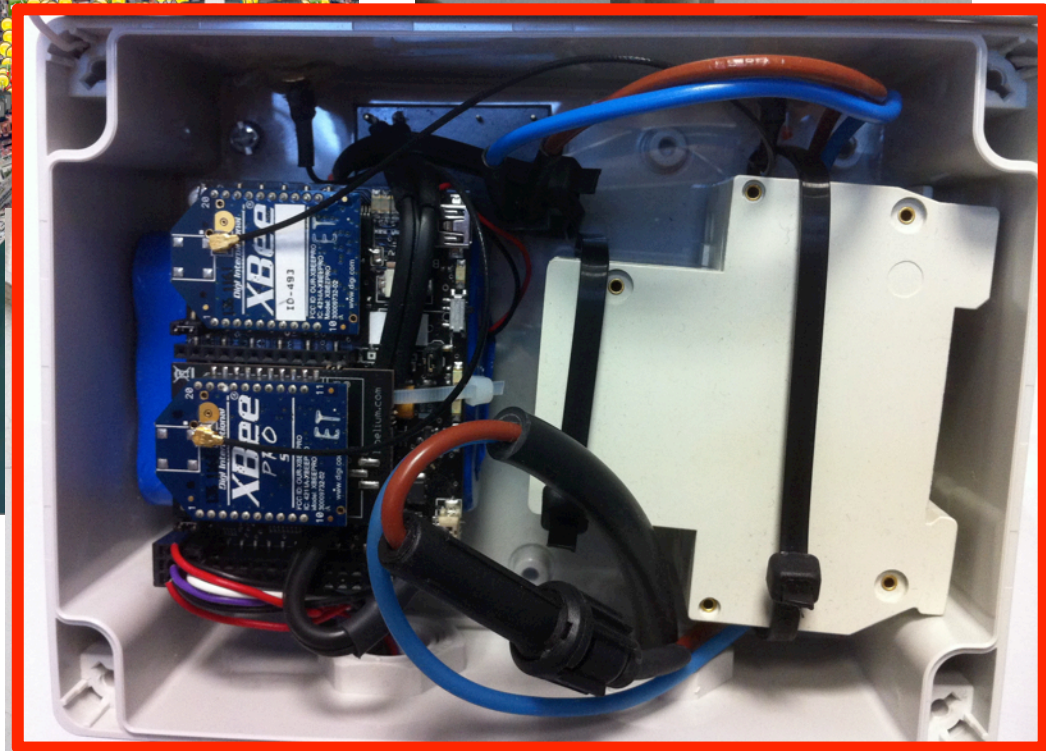
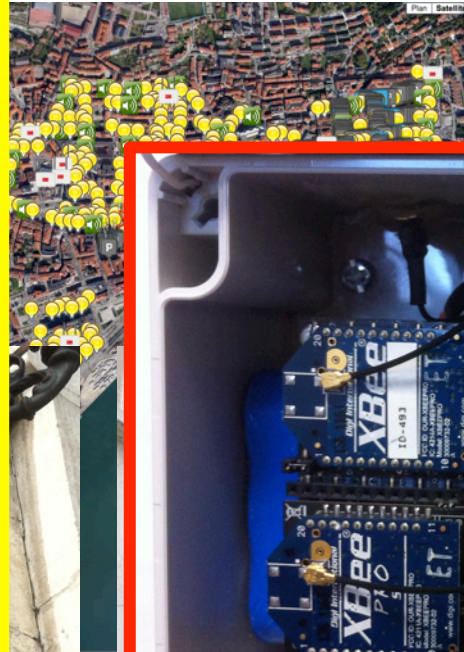
EAR-IT ON SMARTSANTANDER



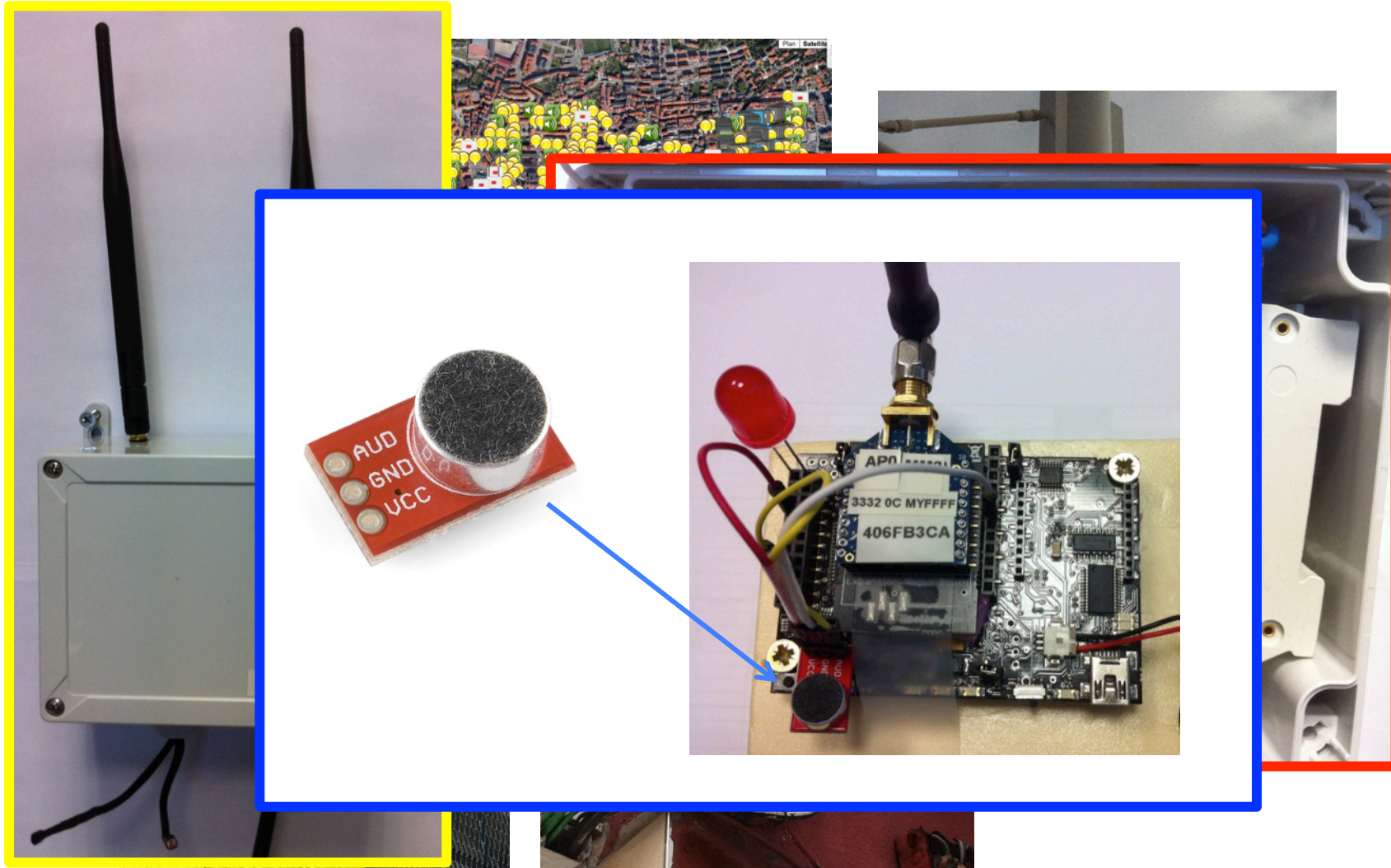
EAR-IT ON SMARTSANTANDER



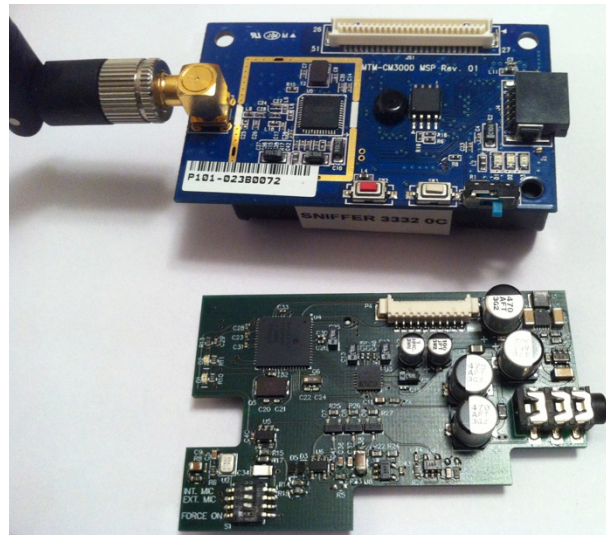
EAR-IT ON SMARTSANTANDER



EAR-IT ON SMARTSANTANDER



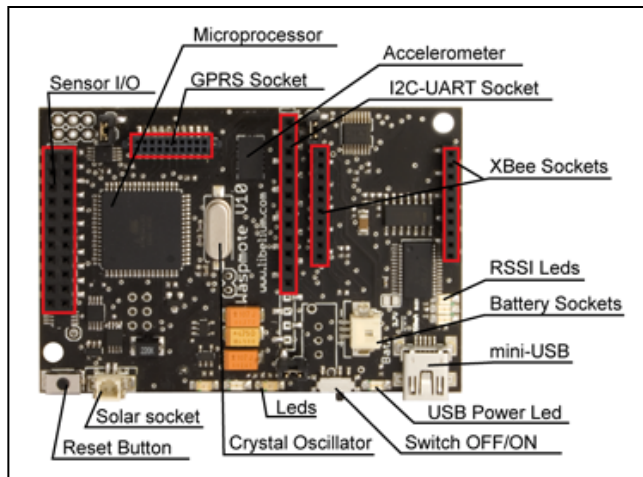
EAR-IT ON HOBNET TEST-BED AT UNIGE



Specially designed audio board by INRIA CAIRNS & Feichter Electronics

dsPIC33 with 8kbps speex real-time encoder

EAR-IT IOT NODE



AdvanticSys CM5000
TelosB-like mote

WaspMote

8MHz Atmega1281
8kB SRAM, 128kB Flash
Xbee radio

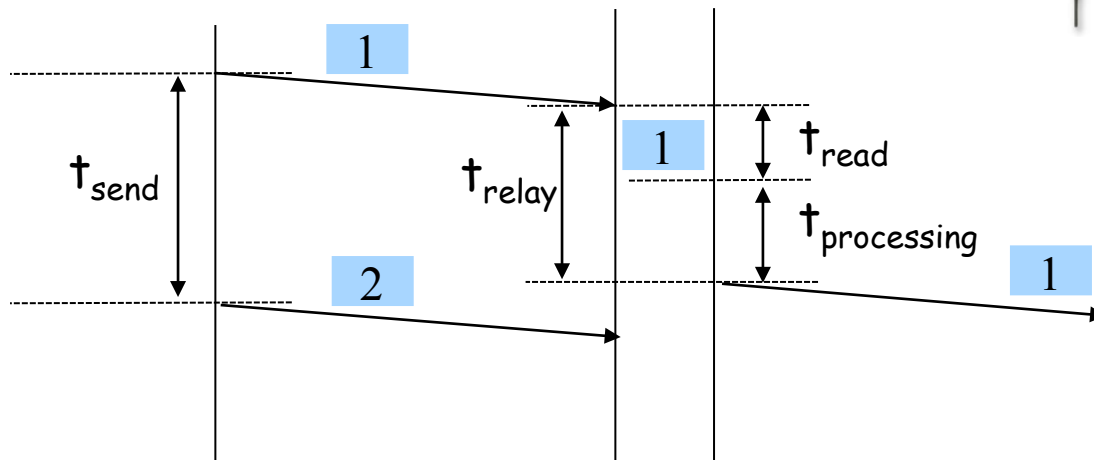
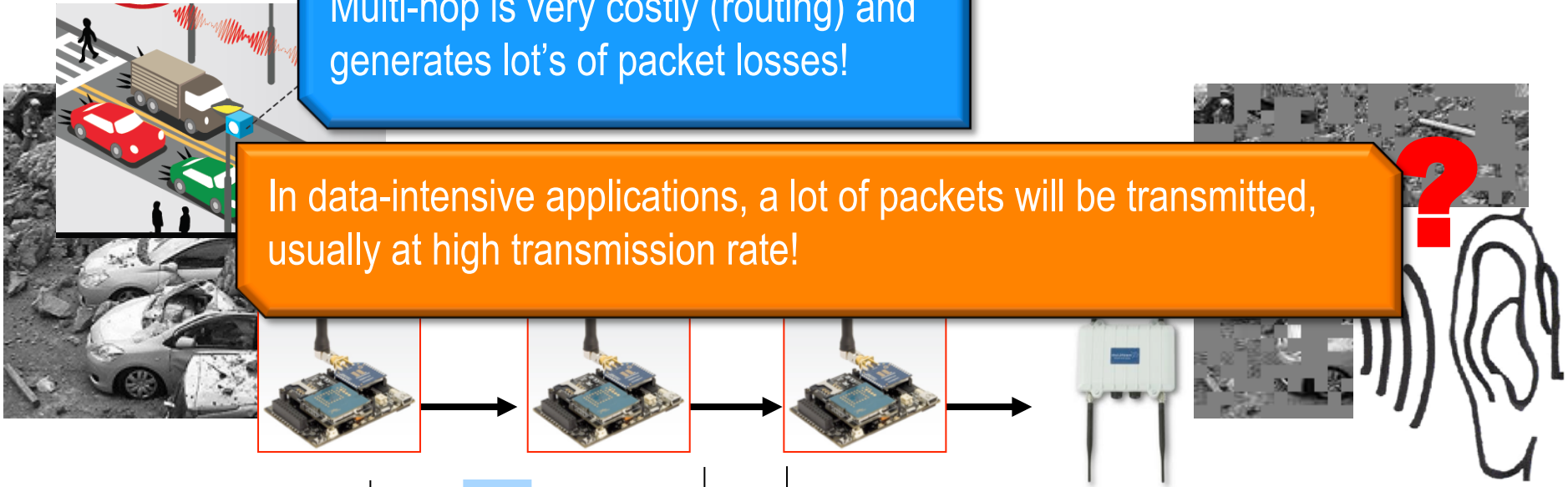


8Mhz MSP430F1611
48K flash, 10K RAM
CC2420 radio

MULTI-HOP PACKET FORWARDING

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

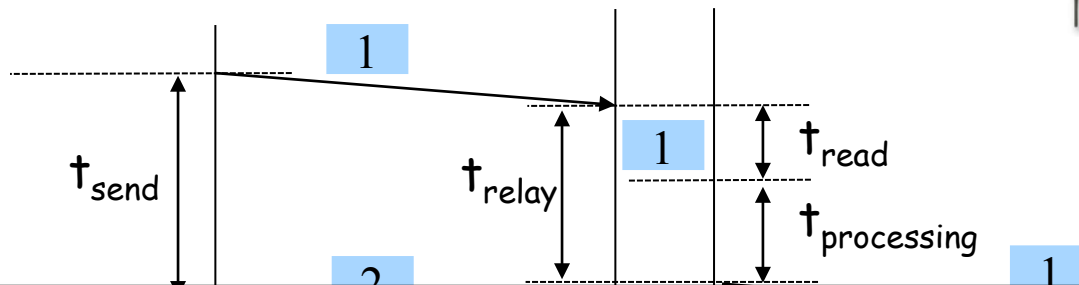
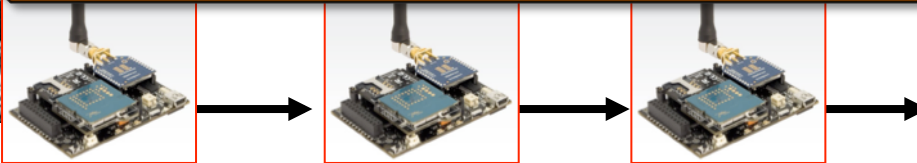
In data-intensive applications, a lot of packets will be transmitted, usually at high transmission rate!



MULTI-HOP PACKET FORWARDING

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

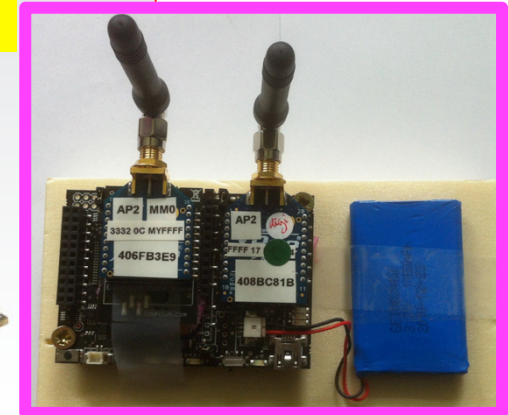
In data-intensive applications, a lot of packets will be transmitted, usually at high transmission rate!



What level of performances can we expect?

MASS-MARKET SENSORS

8MHz Atmega1281
8kB SRAM, 128kB Flash
Xbee radio



LIBELIUM WASPMOTE

COST:
~80€



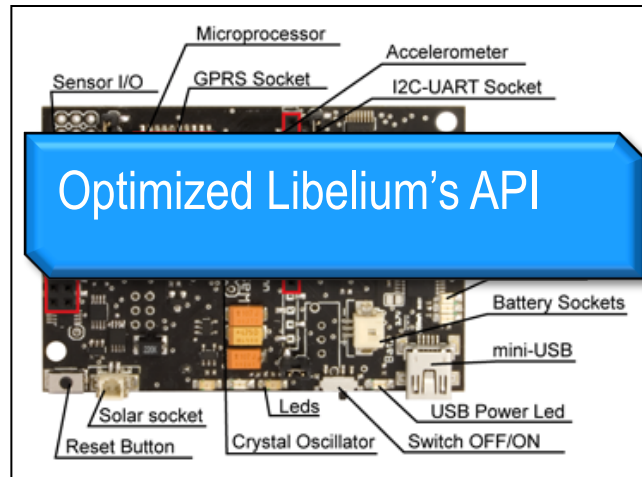
ARDUINO MEGA2560

16MHz Atmega1281
8kB SRAM, 128kB Flash
Xbee radio



SENSOR'S HW&SW

LIBELIUM WASPMOTE



ARDUINO MEGA2560

UART-based connection to micro-controller

Default speed is usually 38400 bauds

Higher baud rate are possible but...

```
WaspXBee802.2_traffic_generator | Waspnote-IDE 02
-----Waspnote XBee 802.15.4 Traffic Generator-----
Version: 0.33
Design: C. Pham
Implementation: C. Pham
*Z10* : set packet size to 20 bytes
*Z130* : increases pkt size from 5 bytes to 100 bytes (or 400 bytes with Libelium API) every 20pkt
*Z200* : set frequency to 1pk/200ms
*TM010A20A0B0C01F* : set destination address to 0010A20A0B0C01F, broadcast by default 000000000000FFFF
*API*/*API* : enable/disable Libelium API with WSPM as mode ID
*PI*/*PI* : enable/disable print sent data
Jun, 14th, 2013, v0.33
adds command string prefix to "/0". All existing command should be prefixed such as: "/0Z10"
March, 19th, 2013, v.0.32
adds support for unsigned long time, fixes wrap around inter-packet time, adds beacon print for long inter-packet time
March, 1st, 2013, v.0.31a
adds support SmartSantander test-bed
Feb, 20th, 2013, v.0.31
adds support for periodic size increase feature
Jan, 15th, 2013, v.0.30
adds reception cad with Digimesh radio module, enable this with USE_UART1, RCV_O0_UART0, RCV_O0_DIGIMESH
and GPS support
Dec, 21st, 2012, v0.2
improves version 0.1 with better timing features and statistics
/* TODO
basic LCD and GPS support need more debugging
// BEGIN of compilation #define statements
// uses advanced timing of the Libelium send API. CAUTION: need modified version of the API
#define SEND_API_TIMING
```

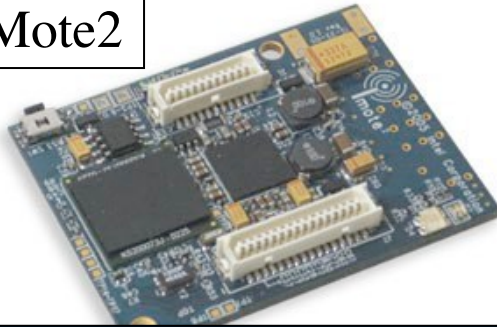
ARDUINO-BASED IDE WITH C++-LIKE LANGUAGE



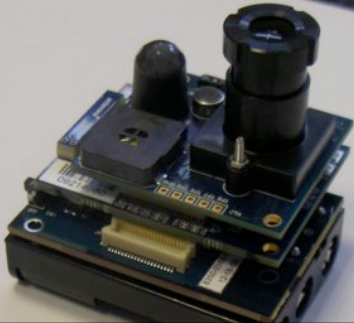
XBEE 802.15.4

« ACADEMIC » SENSORS

iMote2



13-416MHz PXA271 Xscale
Wireless MMX DSP
256kB SRAM, 32MB Flash,
32MB SDRAM
CC2420 radio



iMote2 with IMB400
multimedia board



MICAz

8MHz Atmega128L
4kB SRAM, 128kB Flash
CC2420 radio



Advanticsys CM5000 & CM3000
TelosB-like mote

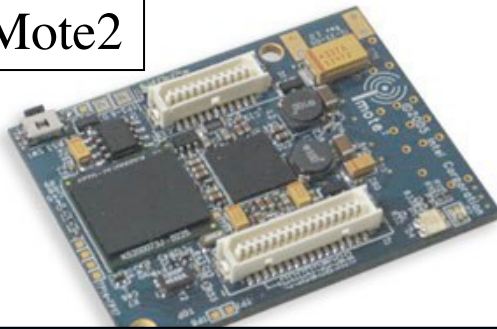


TelosB

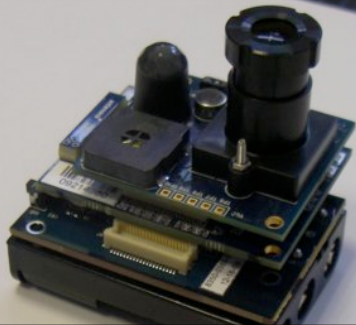
8Mhz MSP430F1611
10K SRAM, 48K flash
CC2420 radio

« ACADEMIC » SENSORS

iMote2



13-416MHz PXA271 Xscale
Wireless MMX DSP
256kB SRAM, 32MB Flash,
32MB SDRAM
CC2420 radio



iMote2 with IMB400
multimedia board



Radio module
CC2420 is
connected
through SPI bus

SPI speed is in
the order of
several
hundredth kbps

TelosB

8MHz Atmega128L
4kB SRAM, 128kB Flash
CC2420 radio

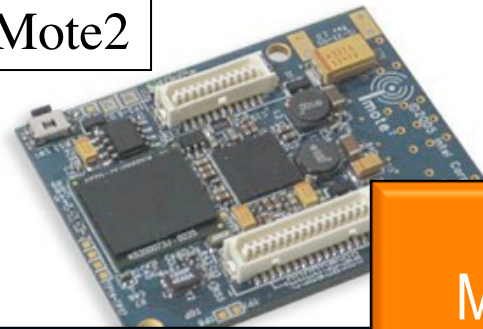


Advanticsys CM5000 & CM3000
TelosB-like mote

8Mhz MSP430F1611
10K SRAM, 48K flash
CC2420 radio

« ACADEMIC » SENSORS

iMote2



8MHz Atmega128L
4kB SRAM, 128kB Flash
CC2420 radio

13-416MHz PXA271
Wireless MMX DSP
256kB SRAM, 32MB
32MB SDRAM
CC2420 radio

Motes are programmed under the
TinyOS operating system & lib

For MicaZ and TelosB we use
TKN154 communication stack

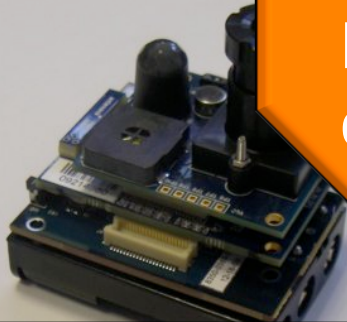
For iMote2 we use IEEE154
communication stack



as CM5000 & CM3000
mote

8Mhz MSP430F1611
10K SRAM, 48K flash
CC2420 radio

iMote2 with IMB400
multimedia board



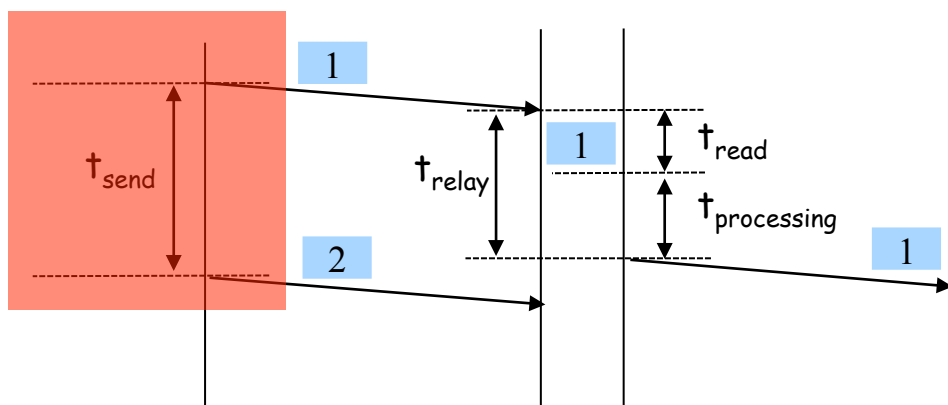
TelosB



COMMUNICATION PERFORMANCES

- ❑ APPLICATION LEVEL PERFORMANCES DEPENDS ON OS, API, HARDWARE ARCHITECTURE
- ❑ USUALLY MUCH LOWER THAN RADIO PERFORMANCES
- ❑ WHAT ARE MINIMUM LATENCIES & MAX. THROUGHPUT?
 - ❑ FOR SENDING?
 - ❑ FOR RECEIVING?
 - ❑ FOR RELAYING?

SENDING PERFORMANCES



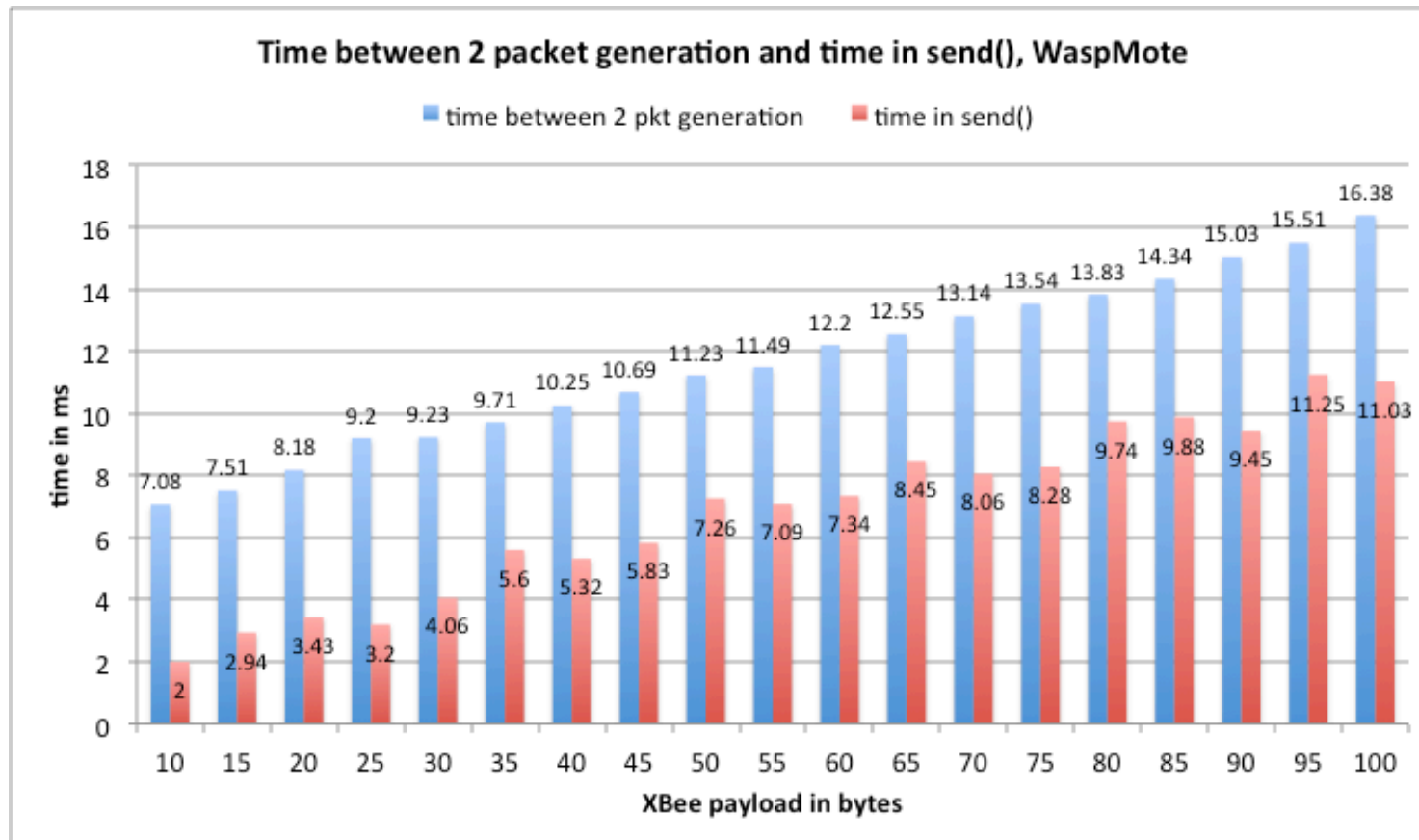
TRAFFIC
GENERATOR

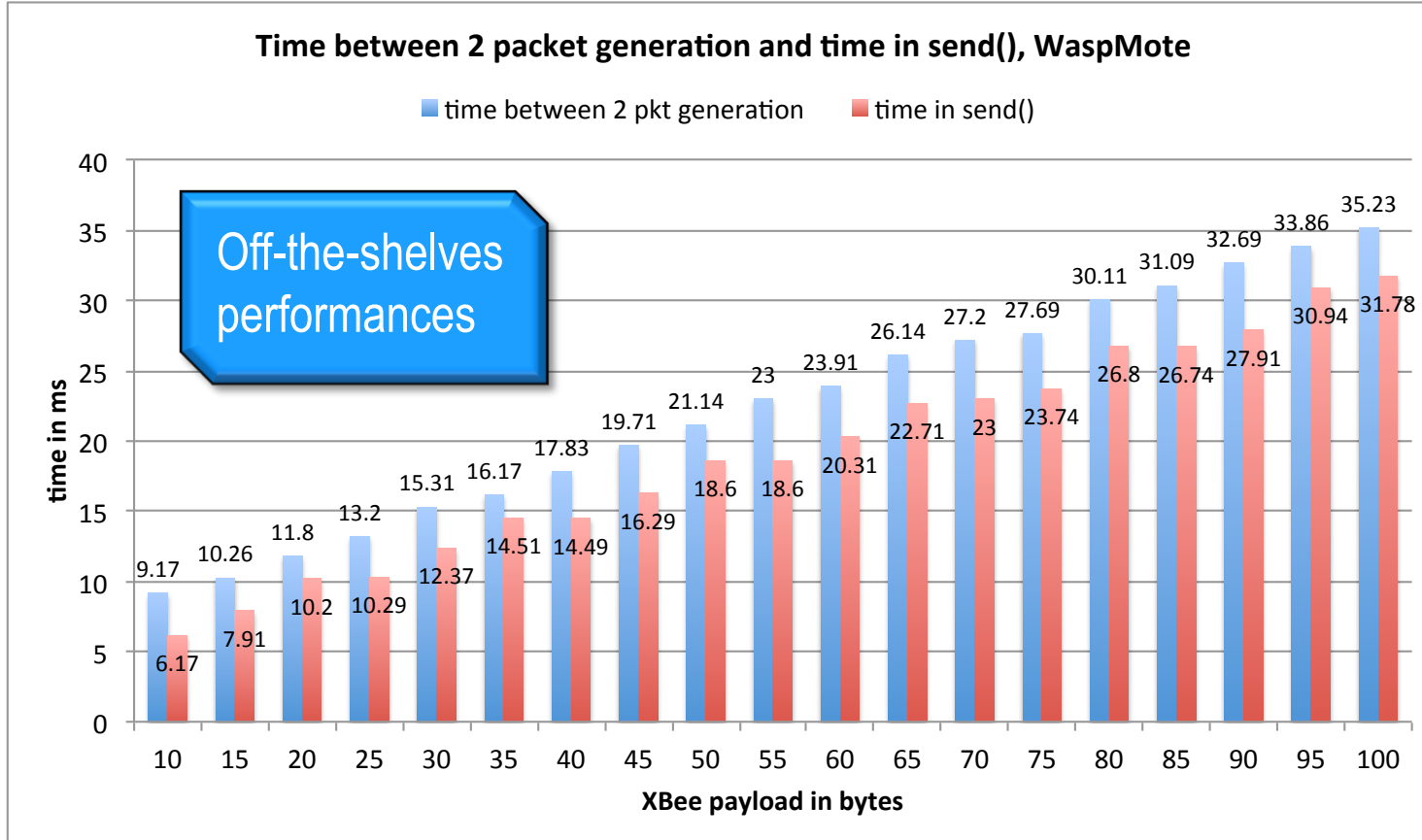
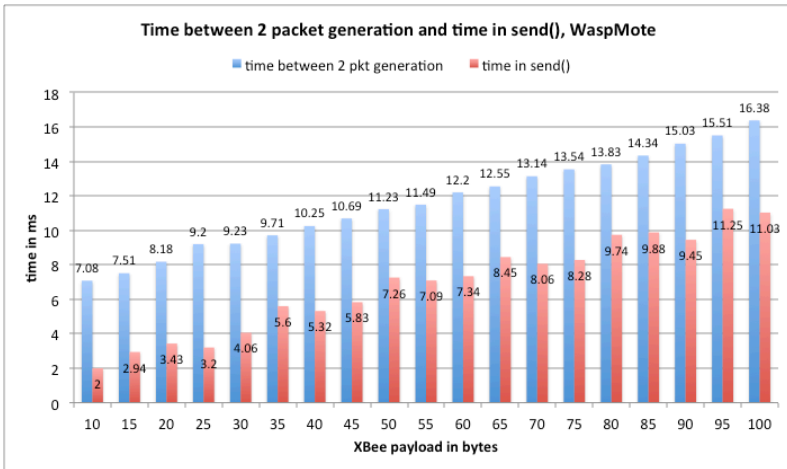
```
void loop() {  
    T0;  
    L0=T0;  
    ...  
    T1;  
    send(buf);  
    T2;  
    ...  
}
```

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

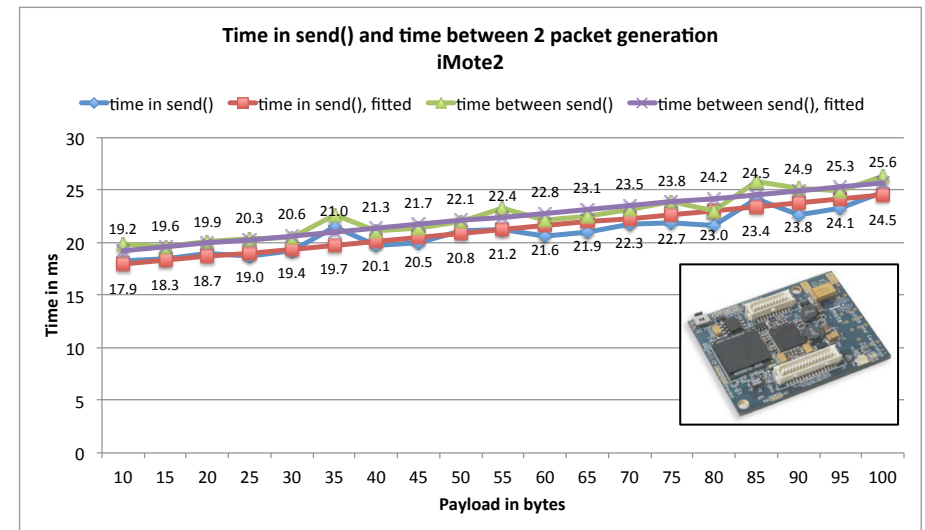
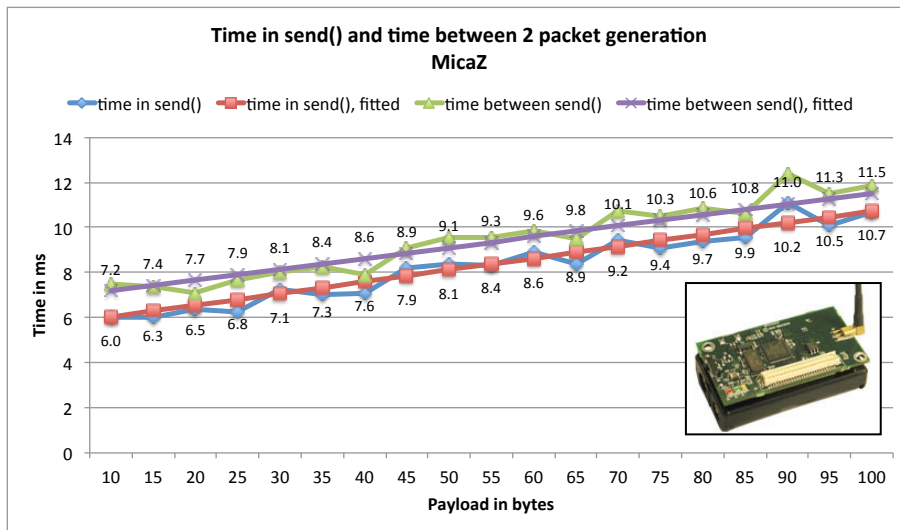
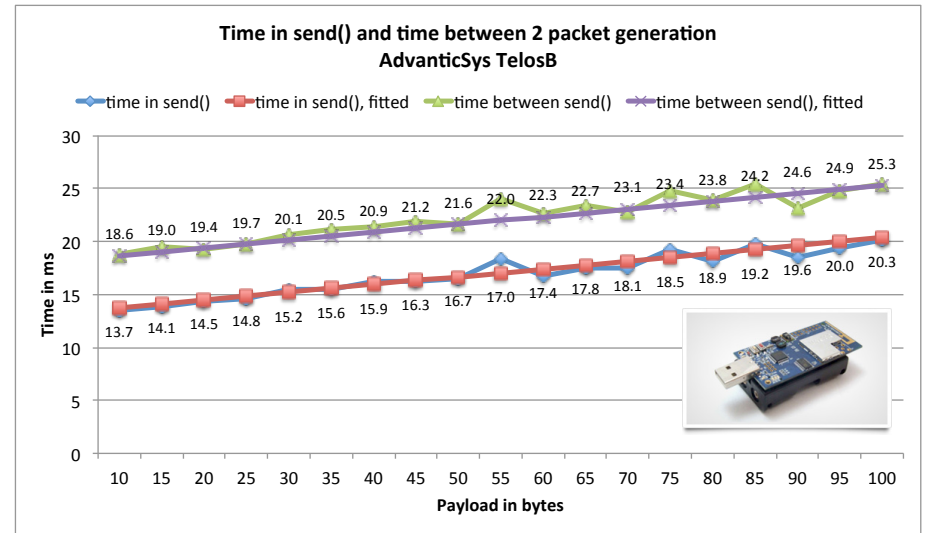
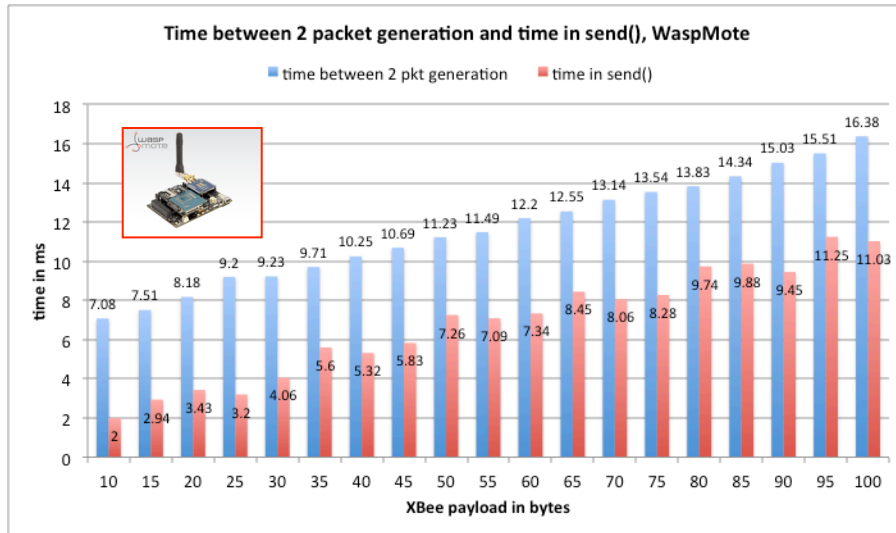
« Time in send() » is T2-T1
« Time between 2 pkt generation » is T0-L0
Time resolution is millisecond
Minimum data manipulation

SENDING PERFORMANCES

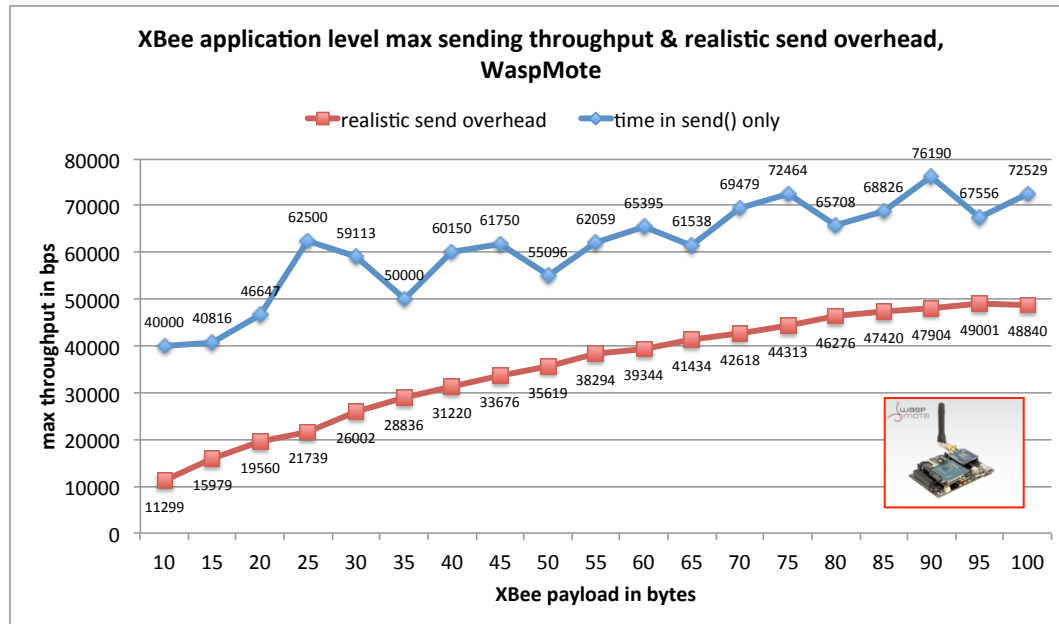




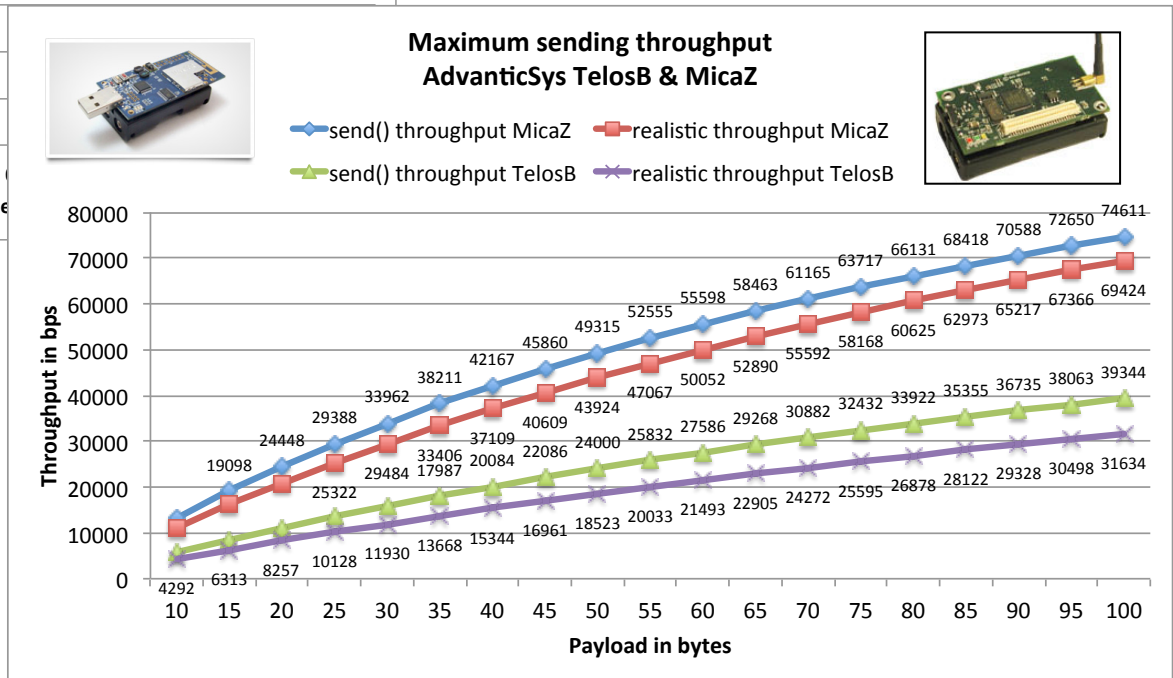
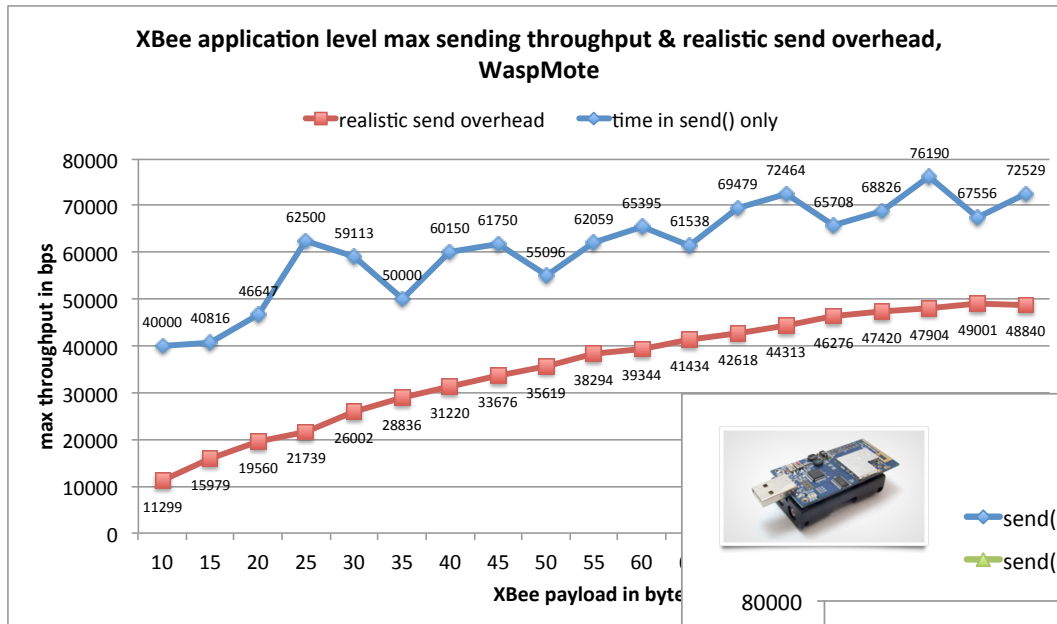
SENDING PERFORMANCES: COMPARISON



MAXIMUM SENDING THROUGHPUT

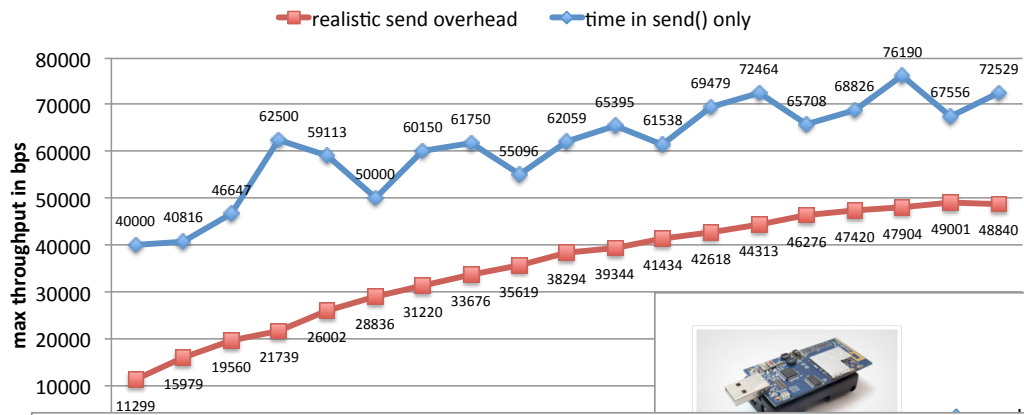


MAXIMUM SENDING THROUGHPUT

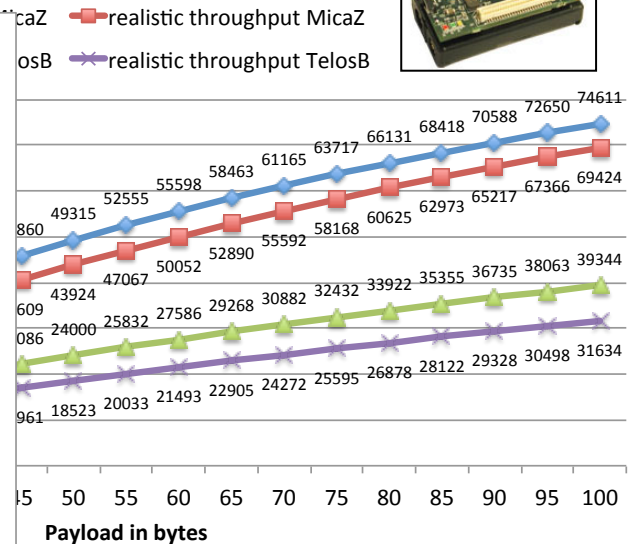
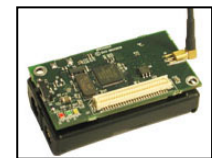


MAXIMUM SENDING THROUGHPUT

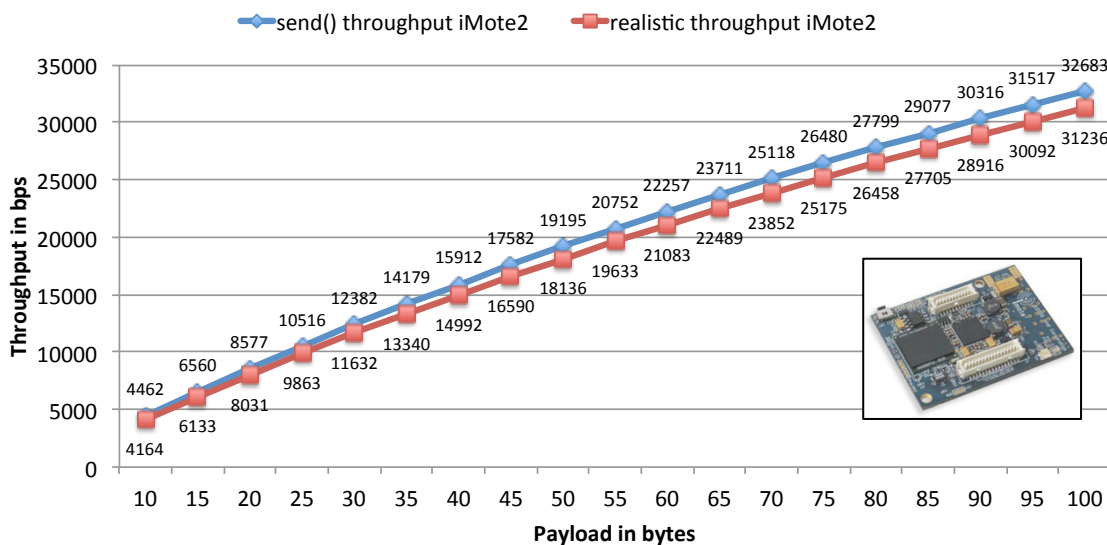
XBee application level max sending throughput & realistic send overhead, WaspMote



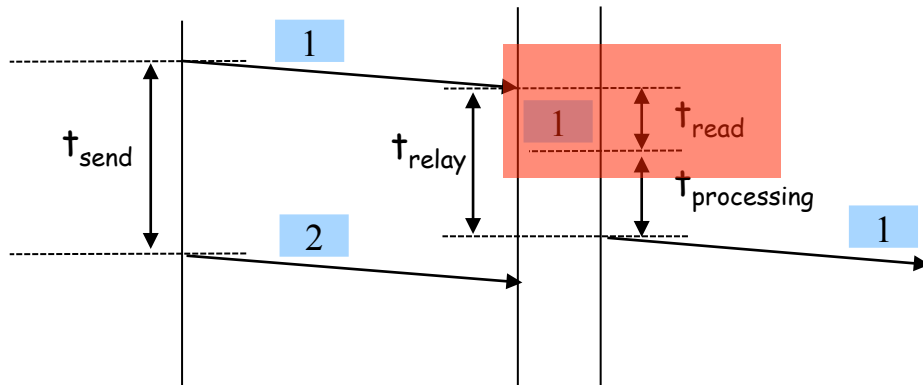
Maximum sending throughput Advanticsys TelosB & MicaZ



Maximum sending throughput iMote2

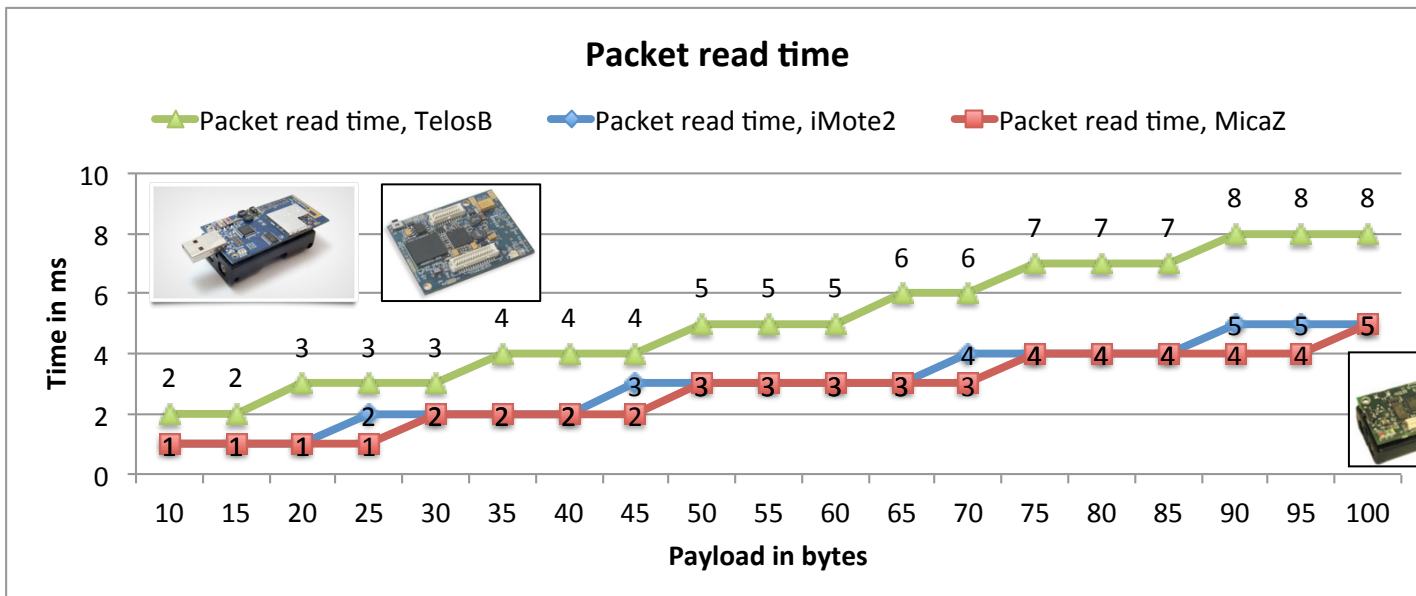
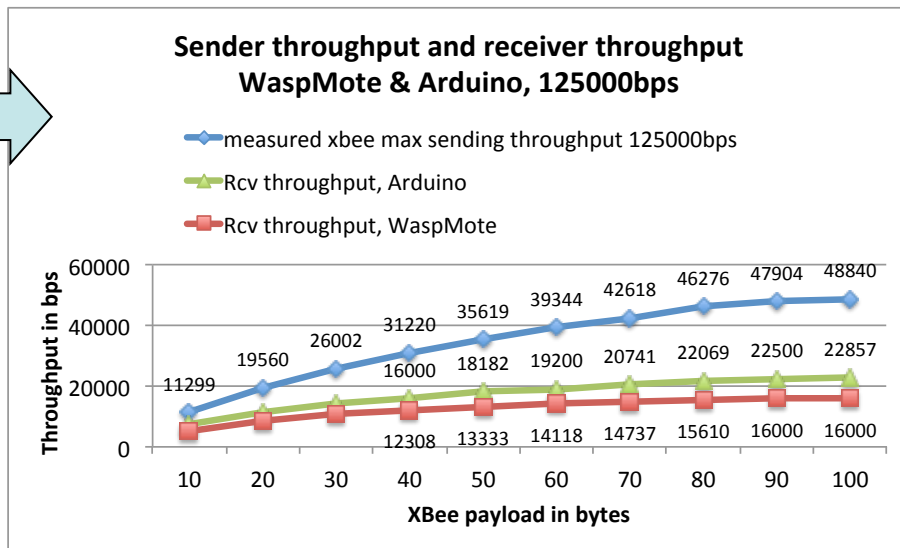
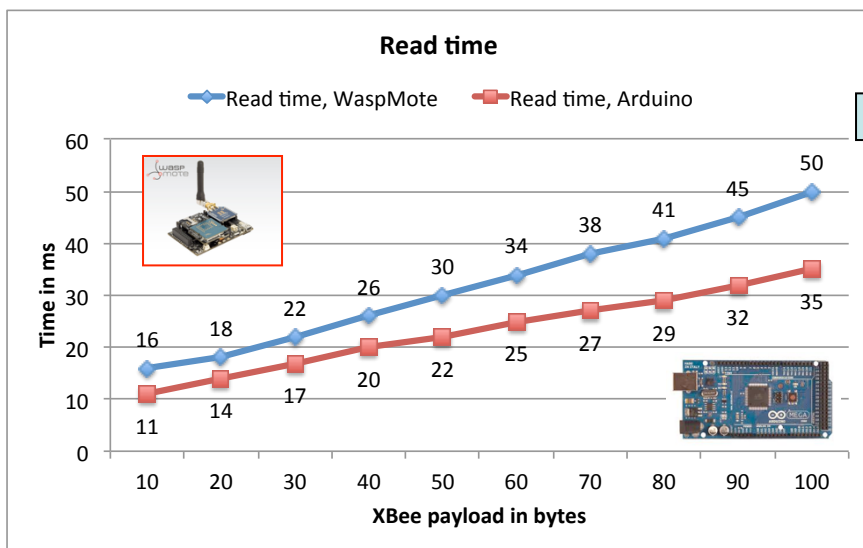


RECEIVE PERFORMANCES

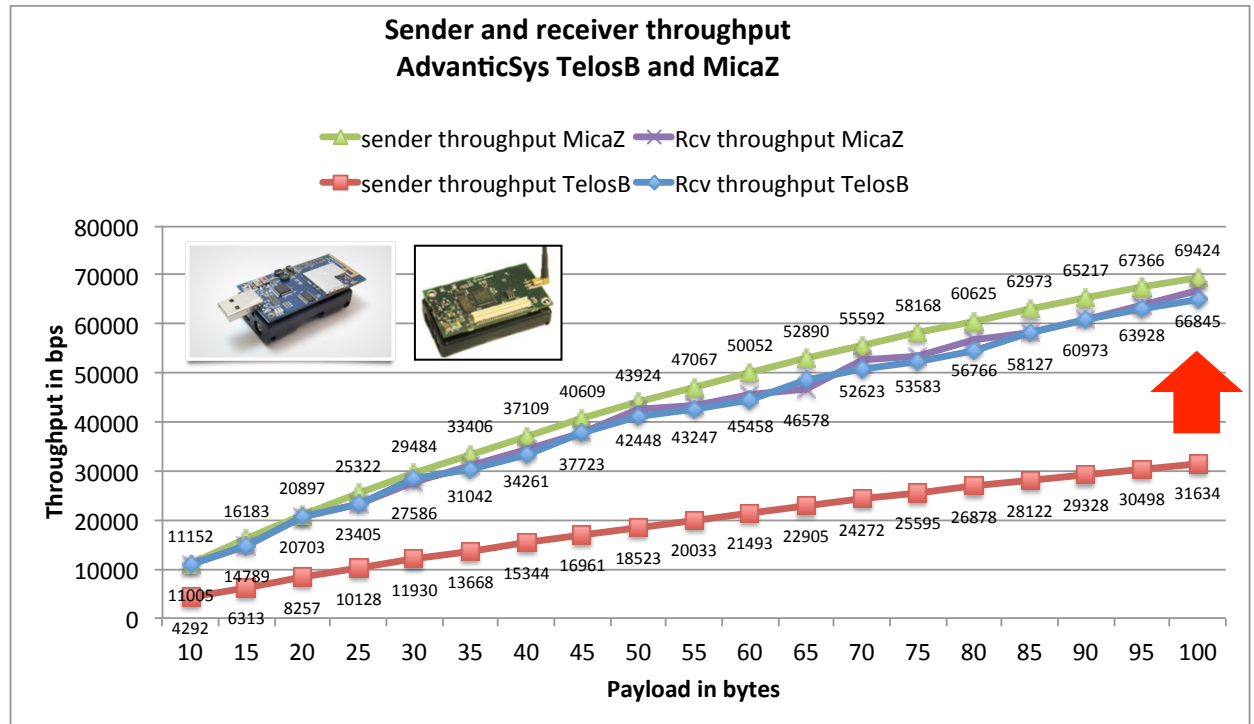


- ❑ AT SENDER SIDE, SEND AS FAST AS POSSIBLE
- ❑ AT RECEIVER SIDE, DETERMINE T_{READ}
- ❑ ... AND ALSO COMPUTE THE MAXIMUM RECEIVE THROUGHPUT PER PACKET SIZE

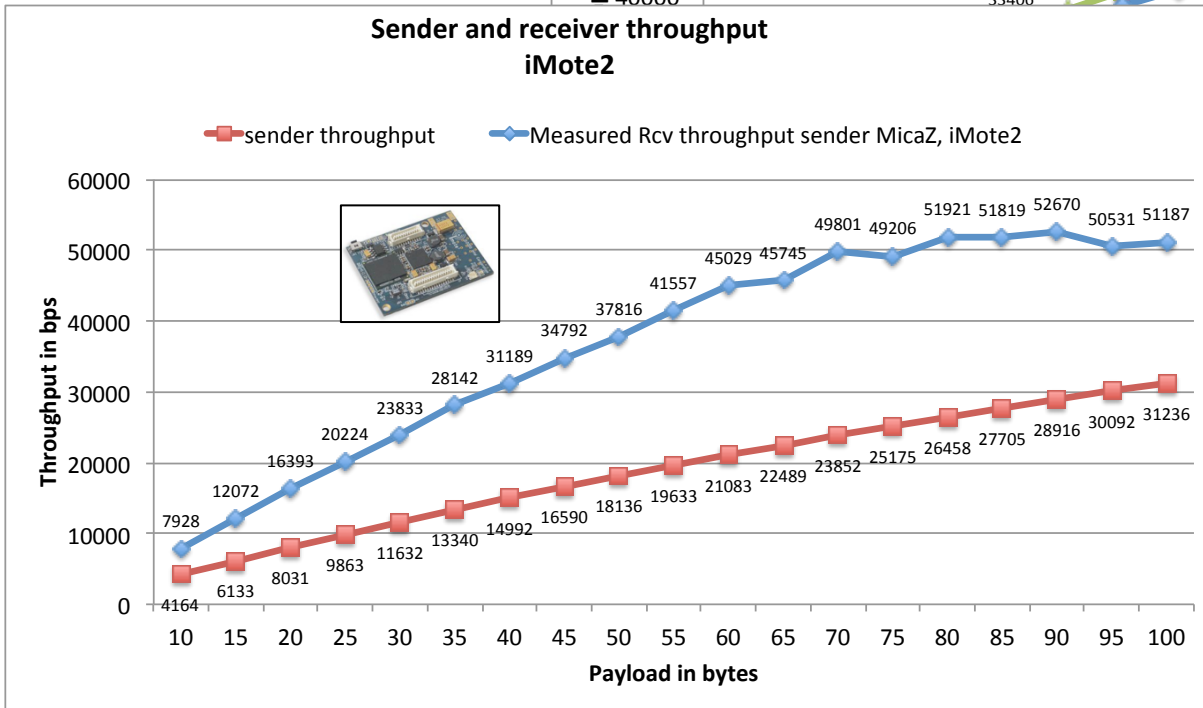
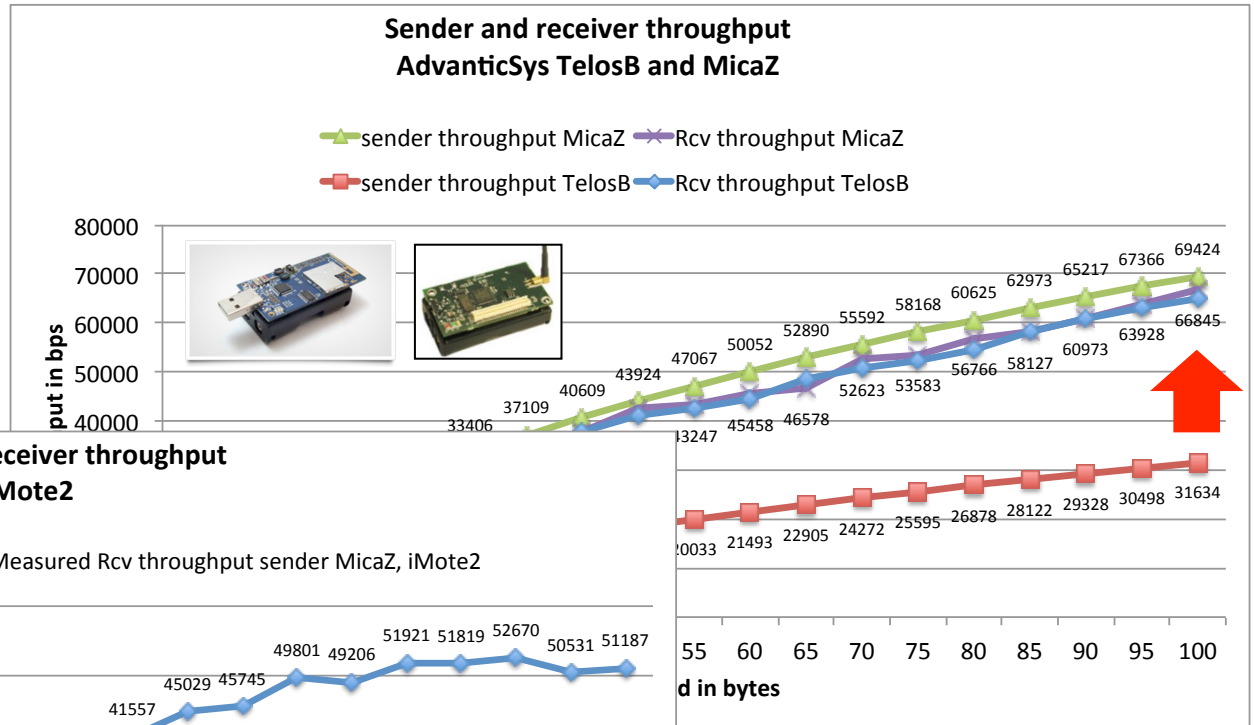
T_{READ} FOR VARIOUS MOTES



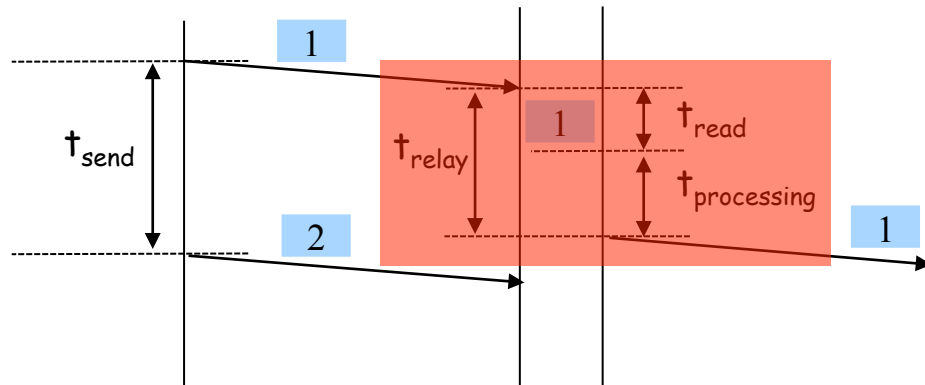
RECEIVER THROUGHPUT



RECEIVER THROUGHPUT

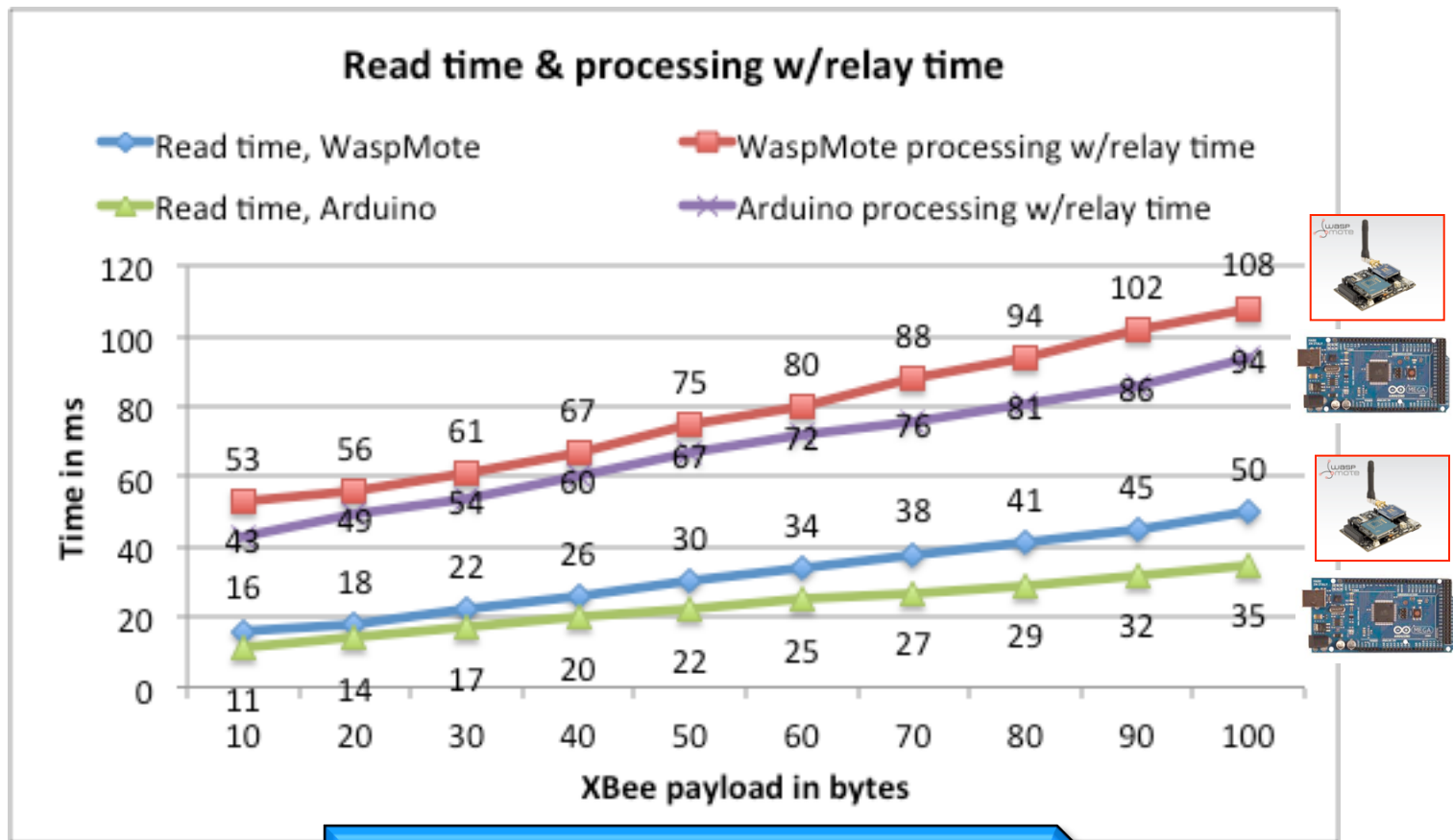


RELAY PERFORMANCES



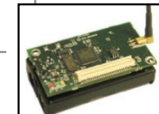
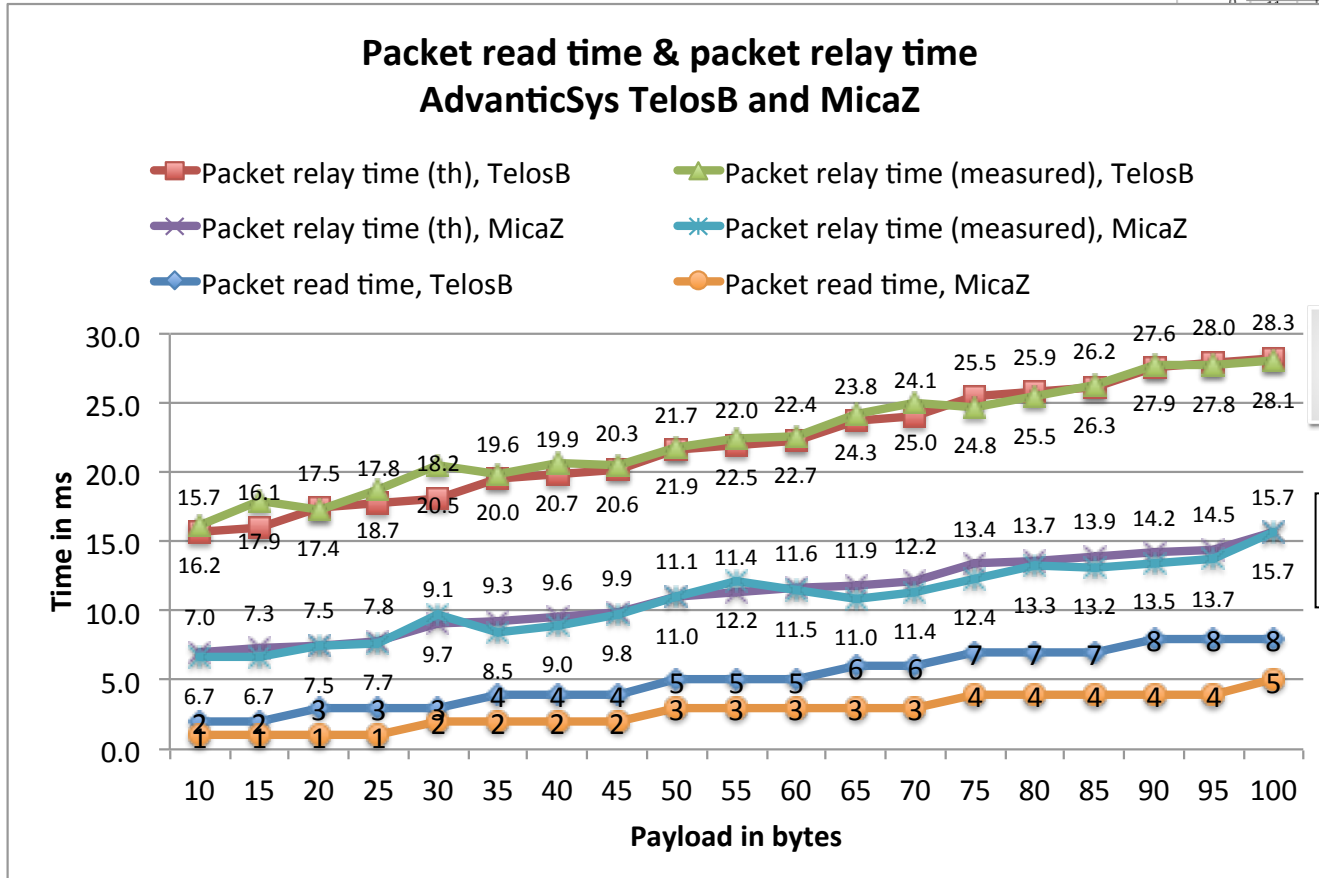
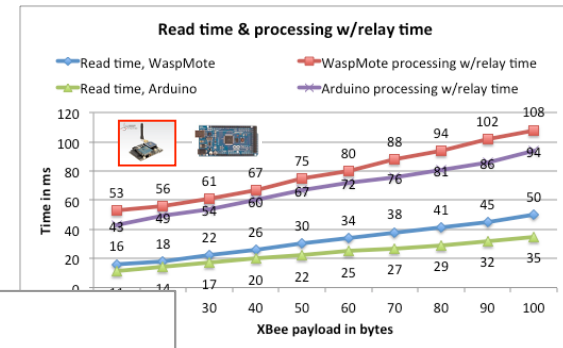
- ❑ RELAYING ARE USUALLY DONE AT APPLICATION-LEVEL (EVEN OS LEVEL IS CONSIDERED APP-LEVEL FOR THE MOTE)
- ❑ RELAYING MEANS:
 - ❑ READ THE PACKET IN MEMORY
 - ❑ SEND THE PACKET TO NEXT HOP

READ TIME AND RELAY TIME

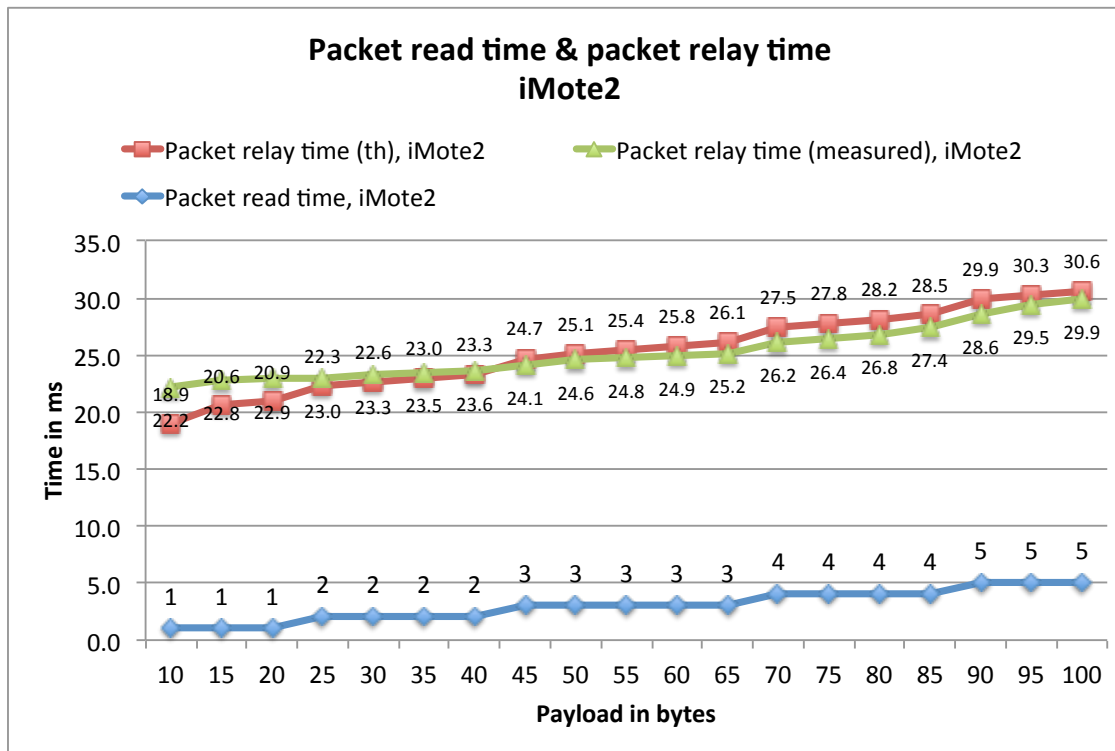
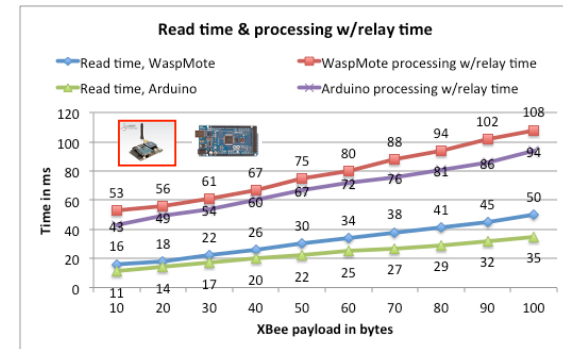
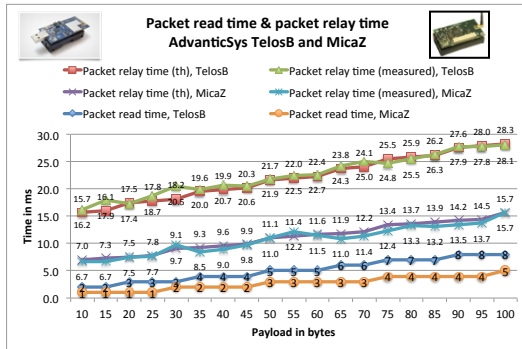


Read time is quite independant from the UART baud rate, but depends on microcontroller frequency

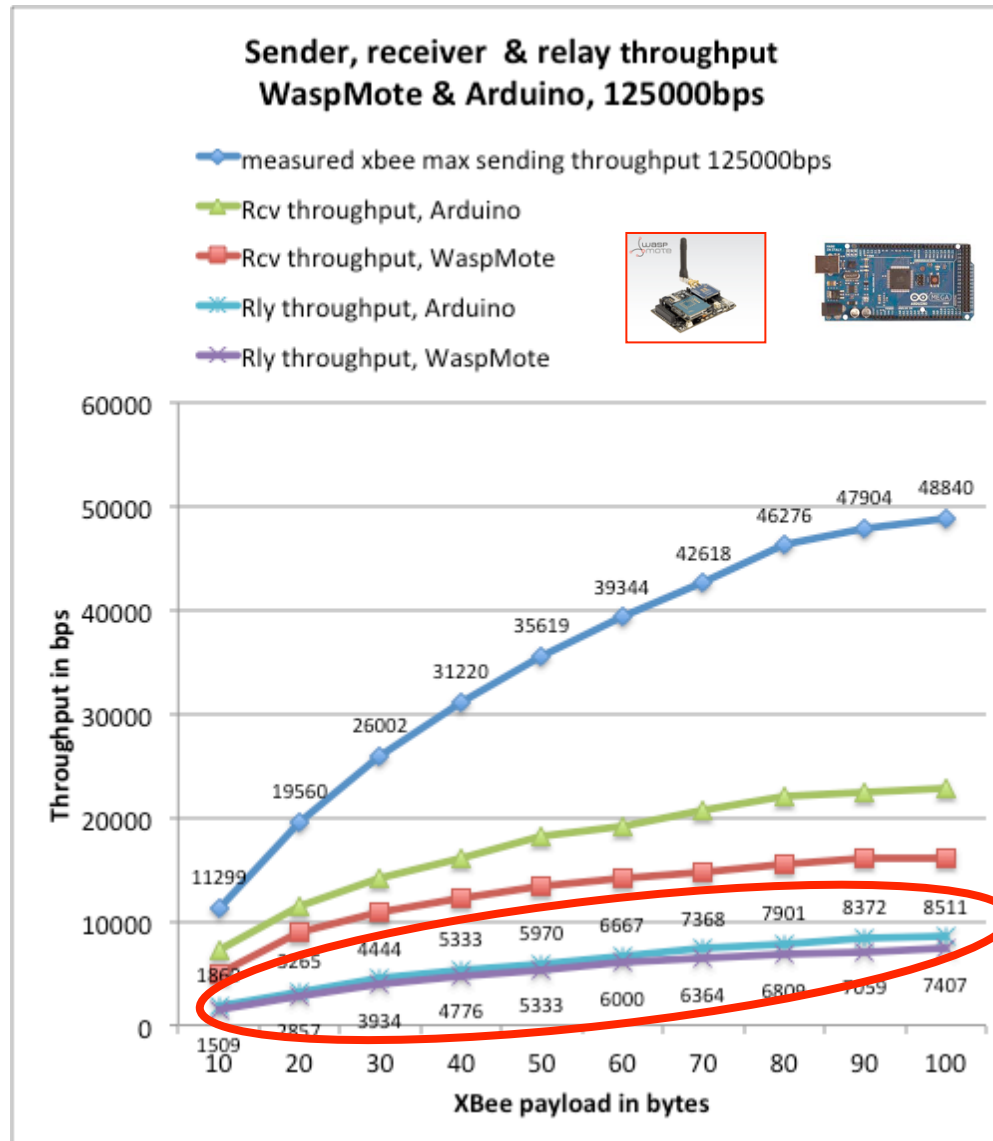
READ TIME AND RELAY TIME



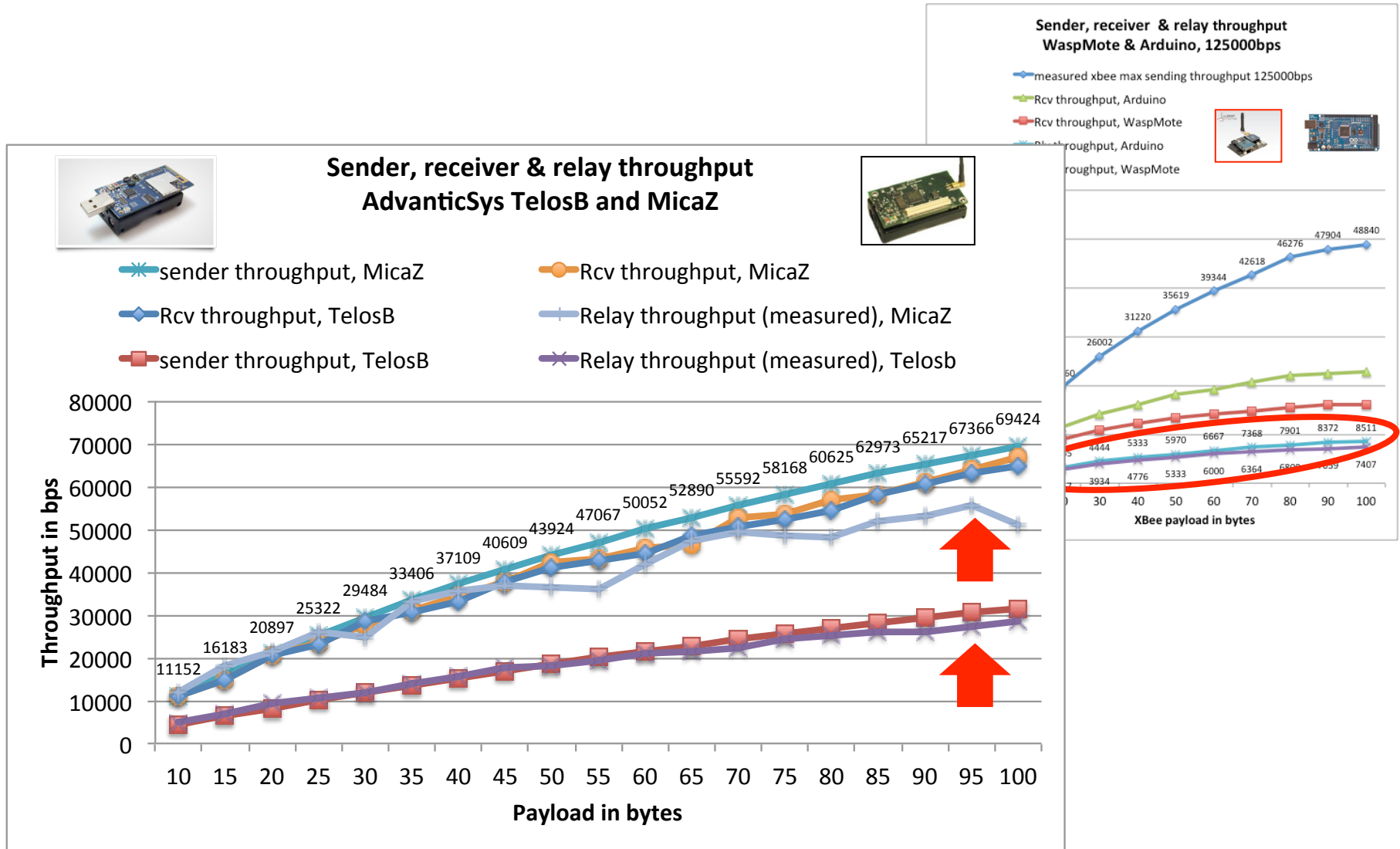
READ TIME AND RELAY TIME



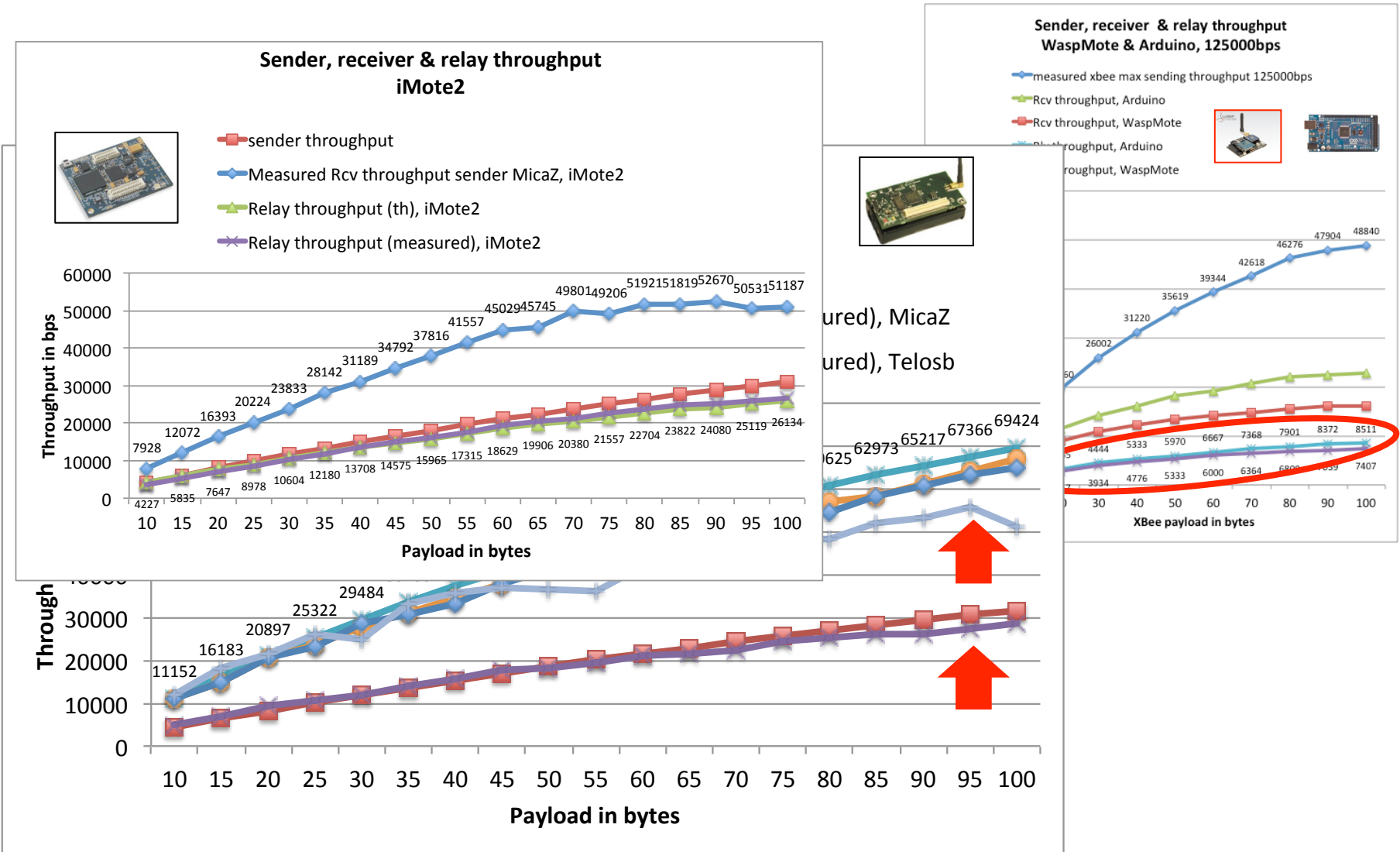
MAXIMUM EXPECTED THROUGHPUT IN MULTI-HOP



MAXIMUM EXPECTED THROUGHPUT IN MULTI-HOP



MAXIMUM EXPECTED THROUGHPUT IN MULTI-HOP



CONCLUSIONS

- ❑ LOOKING AT APP-LEVEL PERFORMANCES, TAKING INTO ACCOUNT OS&API OVERHEADS
- ❑ IN MULTI-HOP COMMUNICATION, RELAY TIME CAN DRAMATICALLY REDUCE THE E2E PERFORMANCES
 - ❑ MASS-MARCKET SENSORS ARE QUITE LIMITED BY READ OVERHEAD
 - ❑ MICAZ APPEARS TO BE THE MOST PERFORMANT PLATFORM
- ❑ CAN BE USED TO BUILD MORE REALISTIC SIMULATION MODEL

BUILDING MORE REALISTIC SIMULATION MODEL

- ❑ DON'T CONSIDER THAT RADIO THROUGHPUT IS AVAILABLE AT APP-LEVEL!
- ❑ USING ONLY TRANSMISSION TIME IS FAR FROM BEING REALISTIC!

The image displays a screenshot of the OMNeT++ simulation environment. The main window shows a network topology with numerous nodes (e.g., node0, node1, node2, etc.) and their connections. A central orange callout box contains the text: "Data-intensive applications (image & audio) really need accurate communication overhead estimations because the number of packets sent is large." Below the callout, there is a code block showing simulation parameters for a node:

```
node59 id 59 px 996 py 304
**Initial_coverage** t=0      pert.coverage 73.152
node 59: initialized with criticality level 0.9 MW Coverage:
node59 id 59 px 996 py 304 alpha 18 vx 1128.74
          by 49.0201      cx 1197.57      cy 133
          Dpv 250 Dpb 262.866 Dpc 262.866
```

In the bottom left corner, there is a small inset image of a hardware device, likely a sensor or a small computer, connected to a network. The simulation interface also shows various control panels and a console window with simulation logs.

SOME LINKS



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

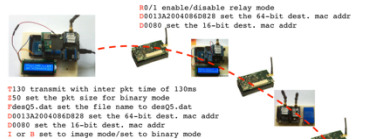


Figure below shows from left to right (source, relay, sink) the 3 main components of our test-bed



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 ([lino.zip](#))
2. code for an Arduino-based relay node, it is actually a sniffer with relay capabilities determined at compilation time ([lino.zip](#))
3. code for at the sink for reading the serial port and decode the 802.15.4 packets received by the XBee module ([L3](#))
compile with `g++ -gxtabs+3 -Wno-write-strings -I/usr/include/SDL -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 ASCII commands to an 802.15.4 devices ([L4](#))
compile with `g++ -Wno-write-strings -o XBeeSendCmd XBeeSendCmd.c -lrt`

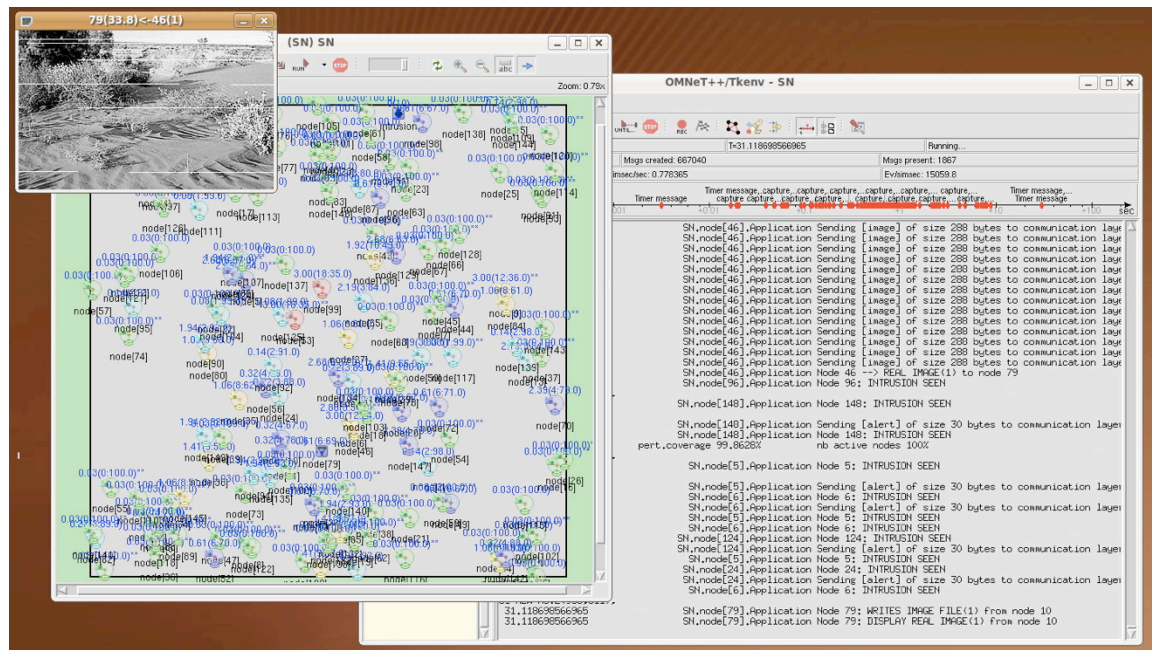


T130 transmit with inter pkt time of 130ms
S50 set the pkt size for binary mode
R500.dat set the file name to r500p.dat
R001A2094084028 set the 44-bit dest. mac addr
D000 set the 14-bit dest. mac addr
1 or 0 set to image mode/set to binary mode

All commands must be prefixed by `! />` and coded separated by `! #`

Example:
`!t130_.../!R!p500.dat#`

XBeeReceive this tool



<http://web.univ-pau.fr/~cpham/WSN-MODEL/wwsn-castalia.html>