FINAL MASTER THESIS

Master of Science in Information and Communication Technologies

Simulation of a sensor network embedded in rail train

Albane Delos

albane.delos@gmail.com

Adviser: Jordi Casademont Serra

UPC - ETSETB

Barcelona, October 2010

Table of contents

Table of figures

Table of graphics

List of tables

Acronyms

AODV	Ad hoc On-Demand Distance Vector
BE	Backoff Exponent
BLE	Battery Life Extension
BO	Beacon Order
CAP	Contention Access Period
CBR	Constant Bit Rate
CCA	Clear Channel Assessment
CFP	Contention-Free Period
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
DDD	Data Display Debugger
FFD	Full Function Device
GDB	GNU DeBugger
GTS	Guaranteed Time Slot
IFS	InterFrame Space or Spacing
LIFS	Long InterFrame Spacing
LR-WPAN	Low Rate Wireless Personal Area Network
MAC	Medium Access Control
MCPS	MAC Common Part Sublayer
MLME	MAC subLayer Management Entity
MPDU	MAC Protocol Data Unit
MSDU	MAC Service Data Unit
PAN	Personal Area Network
PHY	PHYsical layer
PIB	PAN Information Base
POS	Personal Operating Space
PPDU	PHY Protocol Data Unit
PSDU	PHY Service Data Unit
RFD	Reduced Function Device
SIFS	Short InterFrame Spacing
SAP	Service Access Point
SO	Superframe Order
SPDU	SSCS Protocol Data Units
SSCS	Service-Specific Convergence Sublayer
WPAN	Wireless Personal Area Network

Introduction

Wireless communications are becoming more and more important in our world. Wireless communication is an alternative to wire communications, often more expensive to deploy due to the extensive number of cables. In some location, it is simply not possible to deploy cables and wireless communications come up handy reducing by far the expenses. We can also see the part of wireless communications in our day to day life, from cellular phone, to computer wifi and even headphones.

One of the other ways to use wireless communications is through sensor networks. Sensors are small devices with small transmission range and low energy consumption. Sensors can be used in environmental monitoring, for example to monitor temperature in a forest and, this way, prevent the apparition of fires.

The standard protocol 802.15.4 defines a physical layer and a MAC layer which are well suited to be used in sensor network such as LR-WPAN (Low Rate Wireless Personal Area Network). This project will focus on the simulation of a sensor network using 802.15.4 protocol deployed on train wheels. Those sensors would be able to retrieve information as temperature or vibration values and transmit them to a more powerful computer able to treat that information.

First of all, I will rapidly describe some simulators which could be used to perform the simulation. By the study of some characteristics, a choice will be drawn.

Then, to ensure the reliability of the simulator, a set of tests will be conducted and the results obtained will be compared to theoretical ones.

Finally, the last part will deal with the simulation of the sensor network on the train. I will explain the protocol itself and then perform some tests of energy consumption and throughput.

1. Previous study: simulators

First of all, in order to find the adequate simulator for the sensor network using protocol 802.15.4, an analysis of several simulators has been done. The simulator has to be reliable and free of use. Of course it also has to implement 802.15.4 as a native feature.

1.1. Glomosim

GloMoSim stands for Global Mobile Information System Simulator [1]. It is available for academic use only.

GloMoSim uses a parallel discrete-event capability provided by Parsec, C-based simulation language developed by the Parallel Computing Laboratory at UCLA, the University of California. [2]

GloMoSim does not implement 802.15.4 and seems not to be supported anymore. The last version available, GloMoSim 2.0, dates back December 2000. It can be downloaded here [3].

1.2. OPNET

OPNET stands for Optimized Network Engineering Tools.

OPNET technologies Inc. is a company founded in 1986 in the USA by Alain Cohen. It provides a range of solutions for network and application performance management. [4]

OPNET is a commercial software. However, free licenses can be obtained for the use in academic research.
To do that, it is required to fill in a form and provide a complete research description.
To renew a license, the user has to create a web page describing how the research with OPNET license is being utilized. The user must also provide progress reports, such as drafts of papers, final papers, lab materials and sample models to OPNET Technologies Inc. The user may also send copies of published papers. Papers must reference OPNET. There is no guarantee that the license will be renewed. [5]
The form for requiring an academic license can be found in [6].

OPNET offers several solutions [7]:

- **Application Performance Management** will ensure that the application will perform effectively in production, that systems have adequate capacity to support them, and that networks that deliver application functionality can meet service level objectives.

  It consists of:
    - ACE Analyst: *Analystics for Networked Applications*
    - ACE Live: *End-User Experience Monitoring & Real-Time Network Analytics*
    - ACE Enterprise Management: Server *Enterprise-Wide Packet Capture*
    - OPNET Panorama: *Real-Time Monitoring and Analytics*
    - IT Guru Systems Planner: *Systems Capacity Management for Enterprises*
- **Network Engineering, operations and Planning** offers capabilities throughout the entire life cycle of network management, leveraging predictive planning and optimization, network audit and change validation and rapid troubleshooting.

  It consists of:
    - Guru Network Planner: *Network Planning and Engineering*
    - SP Guru Transport Planner: *Transport Network Planning and Engineering*
    - Net Mapper: *Automated Up-to-Date Network Diagramming*
    - Sentinel: *Network Audit, Security and Policy-compliance*
    - OPNET nCompass
- **Network R&D** enables technology innovation and accelerate network protocol and device R&D combining high-fidelity models with industry-leading scalable simulation technologies.

  It consists of:
    - OPNET Modeler: *Accelerating Network R&D*
    - OPNET Modeler Wireless Suite: *Wireless Network Modeling and Simulation*
    - OPNET Modeler Wireless Suite for Defense: *Modeling and Simulation for Defense Communications*

The latest solution: OPNET Modeler is the one interesting in our case.

OPNET provides three hierarchic models: network domain, node domain and process domain. See Figure 1.

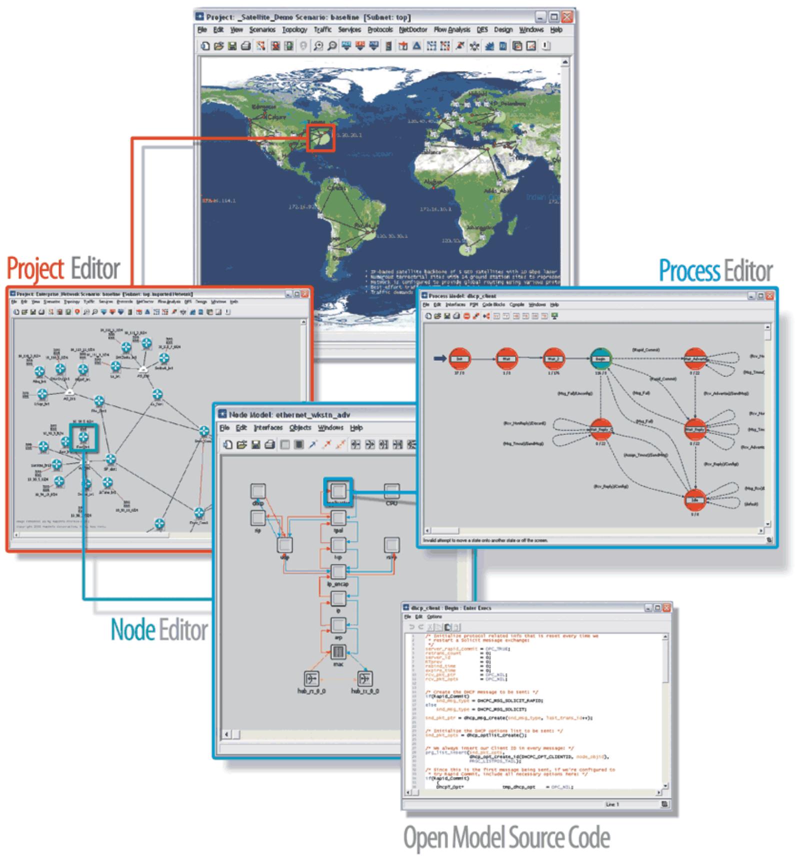


Figure 1 - OPNET Modeler's hierarchical models

<u>Network level/Project Editor:</u>

It is the highest level of OPNET hierarchy. It allows defining the network topology, installing routers, hosts, switches etc.

<u>Node domain/Node Editor:</u>

Node domain enables defining the constitution of nodes. A model is constituted by blocks called modules. Some models can't be programmed and others are entirely programmable. It is described with a Finite State Machine

<u>Process domain:</u>

It is where the role of each module is defined.

OPNET provides a full documentation and a debugger is included in the software.

1.3. OMNeT++

OMNet++ [8] is a component-based, modular and open-architecture discrete event network simulator. It is written in C++ with an Eclipse-based IDE and a graphical runtime environment.

It is popular in academia for its extensibility and plentiful online documentation. It is free for academic and non-profit use.

OMNeT++ can run under Linux, MAC OS or Windows. Its last version is OMNeT++ 4.1 and was released in June 2010. It can be downloaded here: [9].

OMNeT++ seems to be easy to use thanks to its graphic interface. OMNet++ currently provides 802.15.4 standard but only for the non-beacon enabled PANs.

1.4. NS2

1.4.1. General description

NS is a discrete event network simulator. It is open source and so can be extended. This simulator is very well-known and therefore there is plenty of documentation as well as a mailing list where help can be asked.

NS2 [10] is built in C++ and provides a simulation interface through OTcl [11], an object-oriented dialect of Tcl.

OTcl has the advantage of being very easy to use. OTcl serves to describe the topology of the network and specifies the parameters. It interfaces with C++ to perform the simulation. OTcl is easy to use and fast to implement but does not run very fast. On the contrary, C++ needs more time for the implementation but is faster during the launch of the simulation. C++ can be used to create new protocols or to alter one which is already implemented in the simulator.

A hierarchy of the C++ classes are available here [12] as well as the methods and parameters of those classes.

NS2 uses a network animator: nam [13]. It is a Tcl/TK based animation tool for viewing network simulation traces and real world packet traces.

The actual latest version of ns2 is ns-2.34 released June, 17th 2009 and there will be no new version. It can be downloaded here: [14].

NS2 can be installed either under Windows or Linux but it was aimed at first to run under Linux.

1.4.2. 802.15.4

A module of 802.15.4 had been created first by Jianliang Zheng and Myung J. Lee from SAIT-CUNY Joint Lab and has been then modified by several researchers [15]. The module has been since included in the installation package of the ns2 simulator.

The 802.15.4 module being first a separated module, the global documentation of ns2 does not include documentation for 802.15.4 module, so, it may be complicated at first to understand how it works. Some documents provided by Jianliang Zheng list some parameters and their usage.

In this module, almost all 802.15.4 standard is implemented.

- Non beacon enabled/ beacon enabled
- Direct/Indirect communication
- Sleep mode
- ACK or no ACK mode
- GTS (Guaranteed Time Slot) is not implemented.

The backoff calculation of the CSMA/CA algorithm used by 802.15.4 uses a uniform distribution. The number of backoff slots is chosen randomly, however, the random sequence is the same for every simulation. That is, if we run more than once the same simulation, we will have exactly the same results.

1.5. NS3

1.5.1. General description

NS3 [16] is intended to eventually replace ns2 simulator. It is not backwards compatible with ns2.

As ns2, ns3 is an open-source project; it has got open mailing lists, bug tracker and wiki.

Ns3 will use and needs participation from the research community in order to improve the simulation credibility. It is written in C++ with optional Python interface. There are no more Otcl scripts. All the implementation of the topology of the networks is done through C++ or Python.

The ns3 model node is thought more like a real computer and has a behaviour closer to it.

Ns3 conforms to standard input/output formats so that other tools can be reused, as Wireshark for instance. It is then easier to use tools we already know than to familiarise with new tools included in the simulator which do the same thing.

NS3 is very well documented. NS3 documentation is maintained using Doxygen which allow seeing the hierarchy of all the classes and how they work [17].

NS3 pays more attention to realism. The Internet nodes are designed to be a more faithful representation of real computers, including the support for key interfaces such as sockets and network devices, multiple interfaces per nodes, use of IP addresses etc.

The architecture supports the incorporation of more open-source networking software such as kernel protocol stacks, routing daemons, and packet trace analyzers, reducing the need to port or rewrite models and tools for simulation.

NS3 will support virtualization of machines.

NS3 will enable testbed integration. It will be possible to emit or consume network packets over real device drivers or LANs. Therefore, the internal representation of packets is network-byte order.

In NS3, it will be possible to control easily all simulation parameters for static objects and to visualize them into a GUI.

The development of NS3 dated back July 1st 2006 and was foreseen to last 4 years.

The actual version is ns-3.9 and has been released on August 20th 2010. Each new version brings new features and protocols implemented. Actually, ns3 focuses on the development of 802.11 protocol as well as Ipv6 and WiMax.

1.5.2. 802.15.4

802.15.4 is not currently implemented and is not to be implemented in the next releases. There are no modules either, developed by other sources on 802.15.4.

The only way to work with 802.15.4 under ns3 for now is to develop ourselves a module. 802.15.4 being a complete protocol with physical and MAC layer, it would be very time consuming.

1.6. Conclusion

Several simulators have been studied. Table 1 sums up the presence or absence of different characteristics important to the project.

	Glomosim	OPNET	OMNet++	NS2	NS3
802.15.4	✗ No	✓ Yes	◇ Not the whole standard	✓ Yes	✗ Not yet
Support	✗ Last version: December 2000	✓ Yes	✓ Yes	✓ Yes	✓ Yes
Free of use	✓ Yes for academic use	◇ Yes but there are heavy counterparts	✓ Yes	✓ Yes	✓ Yes
User friendly		✓ Graphic UI	✓ Graphic UI	✗ No graphic UI	✗ No graphic UI

Table 1 - Comparison of several simulators

Glomosim is not at all fitted for this project, it does not have 802.15.4 implemented and is not anymore supported either.

OMNeT++ is a very good software with a lot of documentation. Its graphical interface makes it more user friendly than others. However, only part of the 802.15.4 standard is implemented, therefore, it reduces its application.

OPNET would be the best choice: a lot of documentation, graphic UI. However, the counterparts for using OPNET freely are too heavy and risky because there is no guarantee the license would be renewed.

NS3 shows itself to be more faithful to reality. Nevertheless, 802.15.4 is not yet implemented.

So, if ns2 reveals itself to be reliable with the 802.15.4 module, it will sure be a designated choice.

2. NS2: environment setup

This part will explain thoroughly how to install and set up the NS2 environment. When not knowing how to begin with the installation, it can be difficult and time consuming. This part should provide all the intel needed.

2.1. Installation

NS2 can be installed either under a Windows environment or a Linux environment. However, NS2 was first meant for Linux, this is why the following installation procedure is the one for installing NS2 under Linux environment

2.1.1. Download and install

The sources of ns2 simulator can be downloaded here [18]. This is a tar.gz package.

The package is the all-in-one package which contains everything that is necessary: ns2, tcl etc.

The package can also be gotten directly through the terminal, typing:

```
$   wget   http://nchc.dl.sourceforge.net/sourceforge/nsnam/ns-allinone-
2.34.tar.gz
```

First of all, some tools have to be installed; they will be later used to install ns2. Depending on the linux distribution, those tools can already be installed, but anyway, it can be done just in case.

```
$ sudo apt-get install autoconf automake g++ libxmu-dev
```

Now, the installation can begin. To execute the following commands, we have to be in the directory where we want to install ns2.

```
$ tar -xzvf ns-allinone-2.34.tar.gz
$ cd ns-allinone-2.34
$ sudo apt-get install build-essential autoconf automake libxmu-dev
$ ./install
```

If the installation did succeed, something like that should be displayed:

```
Please    put    /your/path/ns-allinone-2.34/bin:/your/path/ns-allinone-
2.34/tcl8.4.18/unix:/your/path/ns-allinone-2.34/tk8.4.18/unix  into  your
PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.
```

```
IMPORTANT NOTICES:

 (1) You MUST put /your/path/ns-allinone-2.34/otcl-1.13, /your/path/ns-
allinone-2.34/lib, into your LD_LIBRARY_PATH environment variable.

If it complains about X libraries, add path to your X libraries into
LD_LIBRARY_PATH.

If you are using csh, you can set it like:
setenv LD_LIBRARY_PATH <paths>

If you are using sh, you can set it like:
export LD_LIBRARY_PATH=<paths>


 (2) You MUST put /your/path/ns-allinone-2.34/tcl8.4.18/library into
your TCL_LIBRARY environmental variable. Otherwise ns/nam will complain
during startup.

After these steps, you can now run the ns validation suite with

cd ns-2.34; ./validate

For trouble shooting, please first read ns problems page

http://www.isi.edu/nsnam/ns/ns-problems.html.  Also  search  the  ns
mailing list archive for related posts.
```

2.1.2. Set the environment variables

After the installation, some environment variables must be set so that ns2 can find all the components needed.

The file .bashrc has to be modified.

```
$ gedit ~/.bashrc
```

The following lines have to be added at the end of the file and /your/path must be replaced by the folder where ns2 has been installed: /home/yourName for instance.

```
# LD_LIBRARY_PATH
OTCL_LIB=/your/path/ns-allinone-2.34/otcl-1.13
NS2_LIB=/your/path/ns-allinone-2.34/lib
X11_LIB=/usr/X11R6/lib
USR_LOCAL_LIB=/usr/local/lib
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL
_LIB

# TCL_LIBRARY
```

```
TCL_LIB=/your/path/ns-allinone-2.34/tcl8.4.18/library
USR_LIB=/usr/lib
export TCL_LIBRARY=$TCL_LIB:$USR_LIB

# PATH
XGRAPH=/your/path/ns-allinone-2.34/bin:/your/path/ns-allinone-
2.34/tcl8.4.18/unix:/your/path/ns-allinone-2.34/tk8.4.18/unix
NS=/your/path/ns-allinone-2.34/ns-2.34/
NAM=/your/path/ns-allinone-2.34/nam-1.14/
PATH=$PATH:$XGRAPH:$NS:$NAM
```

Then the changes are committed with this command:

```
$ source ~/.bashrc
```

To test if everything is all right:

```
$ ns
```

% should appear in the terminal. exit stops the program.

There is a validation suite to test ns2 but this can take more than one hour and is not really necessary. If the beginning of the simulation works, the program should work properly.

```
$ cd ns-2.34
$ ./validate
```

2.1.3. Troubleshooting

If the following message is displayed while starting ns2:

```
The program 'ns' is currently not installed. You can install it by
typing:
sudo apt-get install host  '''(DO NOT do this step)'''
Make sure you have the 'universe' component enabled
bash: ns: command not found
```

It is either because the environment variables are not set properly or this is still the same terminal used for installation. Sometimes, a change of terminal is necessary.

2.2. Compiling/debugging

2.2.1. In a terminal

To compile the ns2 project from a terminal:

```
./configure
make
```

Configure permits to update the Makefile. Indeed, if some changes in the Makefile have to be done, they have to be made in the file Makefile.in and not Makefile. Configure commits the changes made in Makefile.in in Makefile.

Compiling the whole project after a clean can take several minutes.

2.2.2. With GDB/DDD

GDB [19] is a powerful debugger with a command-line interface. It may seem bothersome at first but it can be a good investment to make.

DDD [20] is a GUI front end for GDB debugger. It permits to see the code and use buttons; however, the command-line interface can still be used.

In order to make GDB/DDD work, we have to enable debug symbols:

Edit the install file

The file *install* is situated in the ns-allinone-2.34 folder.

In the tcl section,

`# Build Tcl8.3.2` (should be around line 421)

We need to add the option `--enable-symbols` to the line:

```
./configure --enable-gcc --disable-shared --prefix=$CUR_PATH ||…
```

so it becomes:

```
./configure  --enable-gcc  --enable-symbols  --disable-shared  --
prefix=$CUR_PATH ||…
```

In the tclcl section:

`# Build tclcl` (should be around line 514)

We need to add `--enable-debug` to the configure line, such that:

```
./configure --with-otcl=../otcl-$OTCLVER ||…
```

becomes

```
./configure --enable-debug --with-otcl=../otcl-$OTCLVER ||…
```

Edit the Makefile.in file

In Makefile.in. Search for `CFLAGS` (should be around line 85), and change it from

```
CFLAGS += $(CCOPT) $(DEFINE)
```

to

```
CFLAGS += -g $(CCOPT) $(DEFINE)
```

Reinstallation/compilation

Now, we have to reinstall ns by typing `./install` in the ns-allinone directory.

Then, recompile with `./configure` and `make`.

Now, we should be able to run ns in ddd or gdb with `gdb ns` or `ddd ns`.

2.2.3. With Eclipse

Eclipse can be a good choice for working with ns2. It is not the lightest IDE but it permits to edit, compile and debug with the same tool. It also provides auto-completion which can be very useful with such a project.

Here is a guide to follow for working with Eclipse

NS2 modification

Edit Makefile:

1. Open "…/ns-allinone-2.34/ns-2.34/Makefile.in"
2. Add those lines anywhere near the top of the file to enable debug symbols
   ```
   CCOPT = -g
   DEFINE = -DNDEBUG
   DEFINE = -DDEBUG
   ```
3. Navigate to …/ns-allinone-2.34/ns-2.34 and run "./configure" and then "make"

Eclipse installation

Download Eclipse IDE for C/C++ developers here: [21].
Extract the archive to the folder where you want to install it.

Adding ns2 as a project

1. Open Eclipse
2. Set the workspace setting as the ns installation path(/home/username/ns-allinone-2.34) by selecting File -> Switch Workspace
3. Choose File -> New -> Project -> C++ Project

Toolchains: Linux GCC

4. Select Project Type as Makefile Project -> Empty C++ Project
5. Enter Project Name as ns-2.34
6. Select “Finish”
7. From the workspace, Selecting the NS-2 Project and choosing Project -> Build All should not give Error.
8. Running the project must open the console with the NS-2 prompt, %

Debug configuration

1. Select Run -> Debug Configurations
2. Choose C/C++ Application. Type in any name (ns should be put by default)
3. Under the Main tab, choose the following (it should be already put by default):
    a. Project as ns-2.34
    b. C/C++ Application as ns. (search project and choose this)
4. Under the Debugger tab, choose GDB Debugger. Uncheck the “Stop on startup at” option.
5. Apply and debug.

Troubleshooting

If you have the following error while compiling:

```
make all
Building file: ../ns_tclsh.cc
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"ns_tclsh.d"
-MT"ns_tclsh.d" -o"ns_tclsh.o" "../ns_tclsh.cc"
Dans le fichier inclus à partir de ../ns_tclsh.cc:28:
../config.h:60:19: attention : tclcl.h : No such file or
directory
../ns_tclsh.cc:34: erreur: ‘Tcl_Interp’ was not declared in this
scope
../ns_tclsh.cc:34: erreur: ‘interp’ was not declared in this
scope
../ns_tclsh.cc:35: erreur: expected ‘,’ or ‘;’ before ‘{’ token
../ns_tclsh.cc: In function ‘int main(int, char**)’:
../ns_tclsh.cc:52: erreur: ‘Tcl_Main’ was not declared in this
scope
make: *** [ns_tclsh.o] Erreur 1
```

Make sure you have chosen **Makefile Project**->Empty C++ Project when creating the project.

3. NS2: 802.15.4 module

In this chapter, a study of the ns2 simulator will be conducted in order to check the reliability of the 802.15.4 module.

3.2. Module architecture

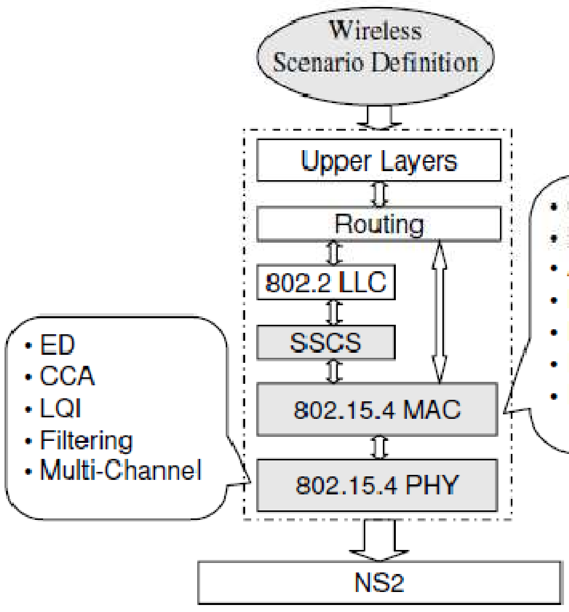


Figure 2 - 802.15.4 module architecture

Figure 2 shows a scheme of 802.15.4 module architecture presented in NS2.

Wireless scenario definition: It selects the routing protocol, defines the network topology and schedules events such as initialization of PAN coordinator, coordinators and devices, and starting/stopping applications. It defines radio-propagation model, antenna model, interface queue, traffic pattern, link error model, link and node failures, superframe structure in beacon enabled mode, radio transmission range, and animation configuration. It is in fact the tcl script part.

SSCS: This is the interface between 802.15.4 MAC and upper layers. It provides a way to access all the MAC primitives, but it can also serve as a wrapper of those primitives.

802.15.4 MAC: This is the main module. It implements the 35 MAC sublayer primitives.

There are two interfaces in the module. One is through the traditional link layer call-back. Another is through IEEE 802.15.4 primitives. By default, it is the traditional link layer which is used.

If we want to use a routing protocol, say AODV, we have to pass through the traditional interface. If we want to use SSCS directly to transmit data, it means that we are not using any routing protocol.

802.15.4 PHY: It implements the 14 PHY primitives

3.3. List of parameters of 802.15.4 in ns2

Here is a list of all 802.15.4 parameters as they appear in ns2 module.

3.3.3. Parameters from the C++ files

Parameter name		Value	File	Comments
PHY layer constants	aMaxPHYPacketSize	127	p802_15_4const.h	Max PSDU size (in bytes) the PHY is able to receive
	aTurnaroundTime	12	p802_15_4const.h	Rx-to-Tx or Tx-to-Rx max turnaround time (in symbol period)
	aCCATime	8	p802_15_4const.h	CCA duration (in symbol period)
PHY_PIB default values	def_phyCurrentChannel	11	p802_15_4const.h	Default value for the channel
	def_phyChannelsSupported	0x07ffffff	p802_15_4const.h	
	def_phyTransmitPower	0	p802_15_4const.h	
	def_phyCCAMode	1	p802_15_4const.h	
MAC sublayer constants	aNumSuperframeSlots	16	p802_15_4const.h	Number of slots contained in a superframe
	aBaseSlotDuration	60	p802_15_4const.h	Duration of a superframe slot of beacon order 0 (in symbol period)
	aMaxBE	5	p802_15_4const.h	Max value of the backoff exponent in the CSMA-CA algorithm

	aMaxBeaconOverhead	75	p802_15_4const.h	Max number of bytes added by the MAC sublayer to the payload of its beacon frame
	aGTSDescPersistenceTime	4	p802_15_4const.h	# of superframes in which a GTS descriptor exists in the beacon frame of the PAN coordinator
	aMaxFrameOverhead	25	p802_15_4const.h	max # of octets added by the MAC sublayer to its payload without security
	aMaxFrameResponseTime	1220	p802_15_4const.h	max # of symbols (or CAP symbols) to wait for a response frame
	aMaxFrameRetries	3	p802_15_4const.h	max # of retries allowed after a transmission failures
	aMaxLostBeacons	4	p802_15_4const.h	max # of consecutive beacons the MAC sublayer can miss without declaring a loss of synchronization
	aMaxSIFSFrameSize	18	p802_15_4const.h	max size of an MPDU, in octets, that can be followed by a SIFS period
	aMinCAPLength	440	p802_15_4const.h	min # of symbols forming the CAP
	aMinLIFSPeriod	40	p802_15_4const.h	min # of symbols forming a LIFS period
	aMinSIFSPeriod	12	p802_15_4const.h	min # of symbols forming a SIFS period
	aResponseWaitTime= 32 * aBaseSuperframeDuration		p802_15_4const.h	max # of symbols a device shall wait for a response command following a request command
	aUnitBackoffPeriod	20	p802_15_4const.h	# of symbols forming the basic time period used by the CSMA-CA algorithm
MAC_PIB default values	def_macAckWaitDuration	54	p802_15_4const.h	22(ack) + 20(backoff slot) + 12(turnaround) The maximum number of symbols to wait for an acknowledgment frame to arrive following a transmitted data frame
	def_macAssociationPermit	false	p802_15_4const.h	Indication of whether a coordinator is currently allowing association

	def_macAutoRequest	true	p802_15_4const.h	Indication of whether a device automatically sends a data request command if its address is listed in the beacon frame
	def_macBattLifeExt	false	p802_15_4const.h	Indication of whether BLE, through the reduction of coordinator receiver operation time during the CAP, is enabled
	def_macBattLifeExtPeriods	6	p802_15_4const.h	In BLE mode, the number of backoff periods during which the receiver is enabled after the IFS following a beacon
	def_macBeaconPayload	“”	p802_15_4const.h	Default beacon payload
	def_macBeaconPayloadLength	0	p802_15_4const.h	Default beacon payload length in octets
	def_macBeaconOrder	15	p802_15_4const.h	Specification of how often the coordinator transmits its beacon. BO=15, the coordinator will not transmit a periodic beacon
	def_macBeaconTxTime	0x	p802_15_4const.h	The time that the device transmitted its last beacon frame, in symbol periods
	def_macCoordExtendedAddress	0xffff	p802_15_4const.h	Default extended address for the coordinator
	def_macCoordShortAddress	0xffff	p802_15_4const.h	Default short address for the coordinator. A value of 0xffff indicates that the value is unknown
	def_macGTSPermit	true	p802_15_4const.h	TRUE if the PAN coordinator is to accept GTS requests
	def_macMaxCSMABackoffs	4	p802_15_4const.h	The maximum number of backoffs the CSMA-CA algorithm will attempt before declaring a channel access failure
	def_macMinBE	3	p802_15_4const.h	Min value of the backoff exponent in the CSMA-CA algorithm
	def_macPANId	0xffff	p802_15_4const.h	The 16-bit identifier of the PAN on which the device is operating. A value of 0xffff signifies that the device is not associated
	def_macPromiscuousMode	false	p802_15_4const.h	Indication of whether the MAC sublayer is in a promiscuous (receive all) mode

	def_macRxOnWhenIdle	false	p802_15_4const.h	Indication of whether the MAC sublayer is to enable its receiver during idle periods
	def_macShortAddress	0xffff	p802_15_4const.h	The 16-bit address that the device uses to communicate in the PAN. A value of 0xffff indicates that the device does not have a short address
	def_macSuperframeOrder	15	p802_15_4const.h	The length of the active portion of the outgoing superframe, including the beacon frame. If SO=15, the superframe will not be active following the beacon
	def_macTransactionPersistenceTime	0x01f4	p802_15_4const.h	The maximum time (in unit periods) that a transaction is stored by a coordinator and indicated in its beacon
	def_macACLEntryDescriptorSet	NULL	p802_15_4const.h	
	def_macACLEntryDescriptorSetSize	0x00	p802_15_4const.h	
	def_macDefaultSecurity	false	p802_15_4const.h	
	def_macACLDefaultSecurityMaterialLength	0x15	p802_15_4const.h	
	def_macDefaultSecurityMaterial	NULL	p802_15_4const.h	
	def_macDefaultSecuritySuite	0x00	p802_15_4const.h	
	def_macSecurityMode	0x00	p802_15_4const.h	
Frequency band and data rate	BR_868M	20	p802_15_4def.h	20 kb/s -- ch 0
	BR_915M	40	p802_15_4def.h	40 kb/s -- ch 1,2,3,...,10
	BR_2_4G	250	p802_15_4def.h	250 kb/s -- ch 11,12,13,...,26
	SR_868M	20	p802_15_4def.h	20 ks/s (symbol rate)
	SR_915M	40	p802_15_4def.h	40 ks/s (symbol rate)
	SR_2_4G	62,5	p802_15_4def.h	62.5 ks/s (symbol rate)
	max_pDelay	100/ 20	p802_15_4def.h	maximum propagation delay
MAC frame control field	defFrmCtrl_Type_Beacon	0x00	p802_15_4field.h	Used internally
	defFrmCtrl_Type_Data	0x04	p802_15_4field.h	Used internally
	defFrmCtrl_Type_Ack	0x02	p802_15_4field.h	Used internally
	defFrmCtrl_Type_MacCmd	0x06	p802_15_4field.h	Used internally
Dest/src addressing mode	defFrmCtrl_AddrModeNone	0x00	p802_15_4field.h	Used internally
	defFrmCtrl_AddrMode16	0x01	p802_15_4field.h	Used internally
	defFrmCtrl_AddrMode64	0x03	p802_15_4field.h	Used internally

Task pending (callback)	TP_mcps_data_request	1	p802_15_4mac.h	Requests the transfer of a data SPDU (ie. MPDU) from a local SSCS entity to a single peer SSCS entity
	TP_mlme_associate_request	2	p802_15_4mac.h	Allows a device to request an association with a coordinator
	TP_mlme_associate_response	3	p802_15_4mac.h	Used to initiate a response to an mlme-association.indication
	TP_mlme_disassociate_request	4	p802_15_4mac.h	Used by an associated device to notify the coordinator of its intent to leave the PAN. It is also used by the coordinator to instruct an associated device to leave the PAN
	TP_mlme_orphan_response	5	p802_15_4mac.h	Allows the next higher layer of a coordinator to respond to the mlme-orphan.indication
	TP_mlme_reset_request	6	p802_15_4mac.h	Allows the next higher layer to request that the MLME performs a reset operation
	TP_mlme_rx_enable_request	7	p802_15_4mac.h	Allows the next higher layer to request that the receiver is either enabled for a finite period of time or disabled
	TP_mlme_scan_request	8	p802_15_4mac.h	Used to initiate a channel scan over a given list of channels. A device can use a channel scan to measure the energy on the channel, search for the coordinator with which it associated, or search for all coordinators transmitting beacon frames within the POS of the scanning device
	TP_mlme_start_request	9	p802_15_4mac.h	Allows the PAN coordinator to initiate a new PAN or to begin using a new superframe configuration. It may also be used by a device already associated with an existing PAN to begin using a new superframe configuration
	TP_mlme_sync_request	10	p802_15_4mac.h	Requests to synchronize with the coordinator by acquiring and, if specified, tracking its beacons
	TP_mlme_poll_request	11	p802_15_4mac.h	Prompts the device to request data from the coordinator
	TP_CCA_csmaca	12	p802_15_4mac.h	

	TP_RX_ON_csmaca	13	p802_15_4mac.h	
	macTxBcnCmdDataHType	1	p802_15_4mac.h	
	macIFSHType	2	p802_15_4mac.h	
	macBackoffBoundType	3	p802_15_4mac.h	
	TxOp_Acked	0x01	p802_15_4mac.h	
	TxOp_GTS	0x02	p802_15_4mac.h	
	TxOp_Indirect	0x04	p802_15_4mac.h	
	TxOp_SecEnabled	0x08	p802_15_4mac.h	
PHY header	defSHR_PreSeq	0x0	p802_15_4pkt.h	
	defSHR_SFD	0xe5	p802_15_4pkt.h	
	defPHY_HEADER_LEN	6	p802_15_4pkt.h	

3.3.4. Parameters in the TCL script

	Parameter name	Value	Comments
node-config	-channel	Channel/WirelessChannel	Channel type
	-propType	Propagation/TwoRayGround	Radio-propagation mode. Also available: FreeSpace and Shadowing
	-phyType	Phy/WirelessPhy/802_15_4	Network interface
	-macType	Mac/802_15_4	Mac layer
	-ifqType	Queue/DropTail/PriQueue	Interface priority queue. Also available: Queue/DropTail
	-llType	LL	Link layer type
	-antType	Antenna/OmniAntenna	Antenna model
	-ifqLen		Max packet in ifq
	-adhocRouting	DumbAgent	Routin protocol. Also available: AODV
	-topoInstance		Topography object
	-agentTrace	ON/OFF	Enable traces at the agent level
	-routerTrace	ON/OFF	Enable traces at the router level
	-macTrace	ON/OFF	Enable traces at mac level

	-movementTrace	OFF	Enable traces for node movement
	-energyModel	“EnergyModel”	Defines the energy model used
	-initialEnergy	1000	Defines the initial energy available in the node (in J)
	-rxPower	35,28e-3	Defines the energy used during a reception (in W)
	-txPower	31,32e-3	Defines the energy used during a transmission (in W)
	-idlePower	712e-6	Defines the energy used while idle (in W)
	-sleepPower	144e-9	Defines the energy used while sleeping (in W)
Traffic	traffic	cbr/poisson/ftp	Type of traffic. Can be cbr, ftp or poisson
CBR	packetSize_		Defines the packet size
	rate_		Bitrate of the source. interval_ and rate_ are mutually exclusive
	interval_		Interval between two packets . interval_ and rate_ are mutually exclusive
	maxpkts_		Max packets to send
	random_	1/0	Specifies whether or not to introduce random noise in the scheduled departure times. Default=off
Phy/Wireless	CsThresh	25m	threshold to determine if the packet is corrupted and will be discarded by the mac layer

	RXThresh	25m	threshold to determine if the packet is detected by the receiver
Mac/802_15_4	wpanCmd ack4data	On/off	Enables or not the use of ack for data packet at the mac level
	wpanCmd verbose	On/off	Enables verbose

3.4. MAC sublayer

This section of the document is here to point out the differences between the MAC sublayer as defined in the 802.15.4 standard [22] and the MAC sublayer which is implemented in the ns2 simulator.

Therefore, it will more be a list of differences rather than really an explication of its mechanism.

For each set of primitives, it will first state the parameters which are or not present in the simulator and in a second part it will quote from the standard, parts which are not implemented or which are different.

3.4.3. General architecture

The MAC sublayer handles all access to the physical radio channel and is responsible for the following tasks:

- Generating network beacons if the device is a coordinator
- Synchronizing to network beacons
- Supporting PAN association and disassociation
- Supporting device security
- Employing the CSMA-CA mechanism for channel access
- Handling and maintaining the GTS mechanism
- Providing a reliable link between two peer MAC entities

The MAC sublayer provides an interface between the SSCS and the PHY. The MAC sublayer includes a management entity called the MLME. This entity provides the service interfaces through which layer management functions may be invoked. The MLME is also responsible for maintaining a database of managed objects pertaining to the MAC sublayer. This database is referred to as the MAC sublayer PIB.

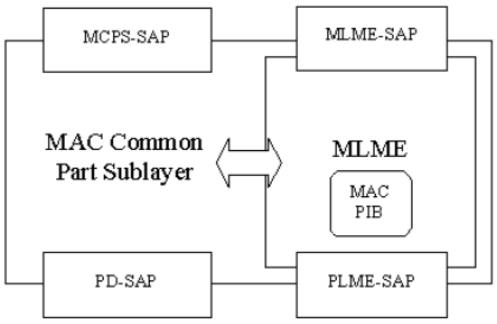


Figure 3 - Scheme of MAC sublayer

Figure 3 shows a scheme of the architecture of MAC sublayer.

The MAC sublayer provides two services, accessed through two SAPs:

- The MAC data service, accessed through the MAC common part layer (MCPS) data SAP (MCPS-SAP)
- The MAC management service, accessed through the MLME-SAP

3.4.4. General notes

Security is not implemented in the simulator. Therefore, the parameter SecurityLevel in the primitives is always set to false and hence the other three security parameters are never used.

Channel pages are not used either.

In the primitive parameters tables, a parameter present in the NS2 architecture is indicated by the ✓ symbol.

3.4.5. MAC data service

The MCPS-SAP supports the transport of SSCS protocol data units (SPDUs) between peer SSCS entities.

MCPS-DATA

The MCPS-DATA primitives request the transfer of a data SPDU from a local SSCS entity to a single peer SSCS entity.

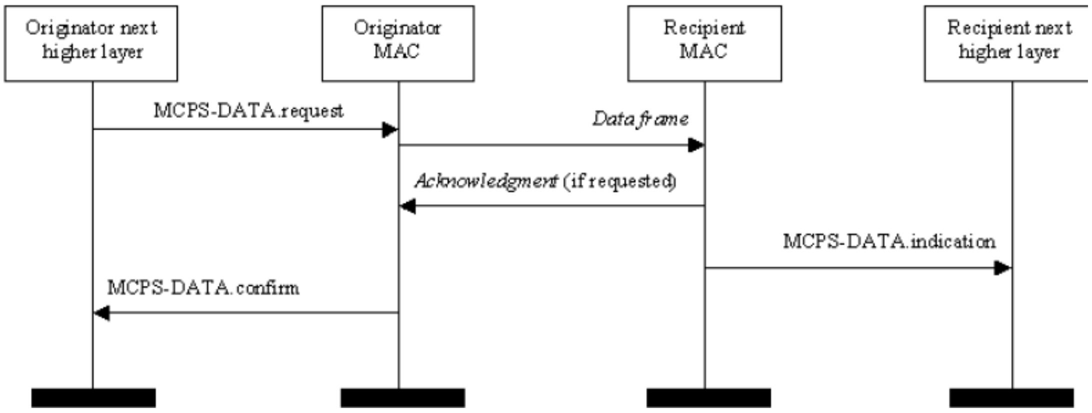


Figure 4 - Sequence diagram of MCPS-DATA primitives

MCPS-DATA primitives

Figure 4 shows the sequence diagram for the exchange of MCPS-DATA primitives.

The MCPS-DATA.request primitive is generated by a local SSCS entity when a data SPDU (i.e. MSDU) is to be transferred to a peer SSCS entity.

The MCPS-DATA.confirm primitive is generated by the MAC sublayer entity in response to a MCPS-DATA.request primitive. The MCPS-DATA.confirm primitive returns a status of either SUCCESS, indicating that the request to transmit was successful, or the appropriate error code.

The MCPS-DATA.indication primitive is generated by the MAC sublayer and issued to the SSCS on receipt of a data frame at the local MAC sublayer entity.

Table 2, Table 3 and Table 4 mention which parameters of each primitive are implemented in NS2.

MCPS-DATA.request	
Simulator	**Standard**
✓	SrcAddrMode
✓	DstAddrMode
✓	DstPANId
✓	DstAddr
✓	msduLength
✓	msdu
✓	msduHandle
✓	TxOptions
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 3 - MCPS-DATA.request parameters

MCPS-DATA.indication	
Simulator	**Standard**
✓	SrcAddrMode
✓	SrcPANId
✓	SrcAddr
✓	DstAddrMode
✓	DstPANId
✓	DstAddr
✓	msduLength
✓	msdu
✓	mpduLinkQuality
	DSN
	Timestamp
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 4 - MCPS-DATA.indication parameters

MCPS-DATA.confirm	
Simulator	**Standard**
✓	msduHandle
✓	status
	Timestamp

Table 2 - MCPS-DATA.confirm parameters

Differences in the behavior

Apart from the security, there are three differences from the standard.

Below are some quotations from the standard.

If both the SrcAddrMode and DstAddrMode parameters are set to 0x00 (i.e., addressing fields omitted), the MAC sublayer will issue the MCPS-DATA.confirm primitive with a status of INVALID_ADDRESS. [23]

If the msduLength parameter is greater than aMaxMACSafePayloadSize, the MAC sublayer will set the Frame Version subfield of the Frame Control field to one. [23]

If there is no capacity to store the transaction, the MAC sublayer will discard the MSDU and issue the MCPS-DATA.confirm primitive with a status of TRANSACTION_OVERFLOW. [23]

Those three tests made on the data SPDU are not implemented in the simulator.

MCPS-PURGE

It is generated by the next higher layer whenever a MSDU is to be purged from the transaction list.

MCPS-PURGE primitives

The MCPS-PURGE.request primitive is generated by the next higher layer whenever a MSDU is to be purged from the transaction queue.

The MCPS-PURGE.confirm primitive is generated by the MAC sublayer entity in response to an MCPS-PURGE.request primitive. The MCPS-PURGE.confirm primitive returns a status of either SUCCESS, indicating that the purge request was successful, or INVALID_HANDLE, indicating an error.

Table 5 and Table 6 mention the parameters present in NS2.

MCPS-PURGE.request	
Simulator	**Standard**
✓	msduHandle

Table 5 - MCPS-PURGE.request parameter

MCPS-PURGE.confirm	
Simulator	**Standard**
✓	msduHandle
✓	status

Table 6 - MCPS-PURGE.confirm parameters

Differences in the behavior

There is no difference between the implementation in the simulator and the standard.

3.4.6. MAC management service

The MLME-SAP allows the transport of management commands between next higher layer and the MLME.

Association

MLME-SAP association primitives define how a device becomes associated with a PAN.

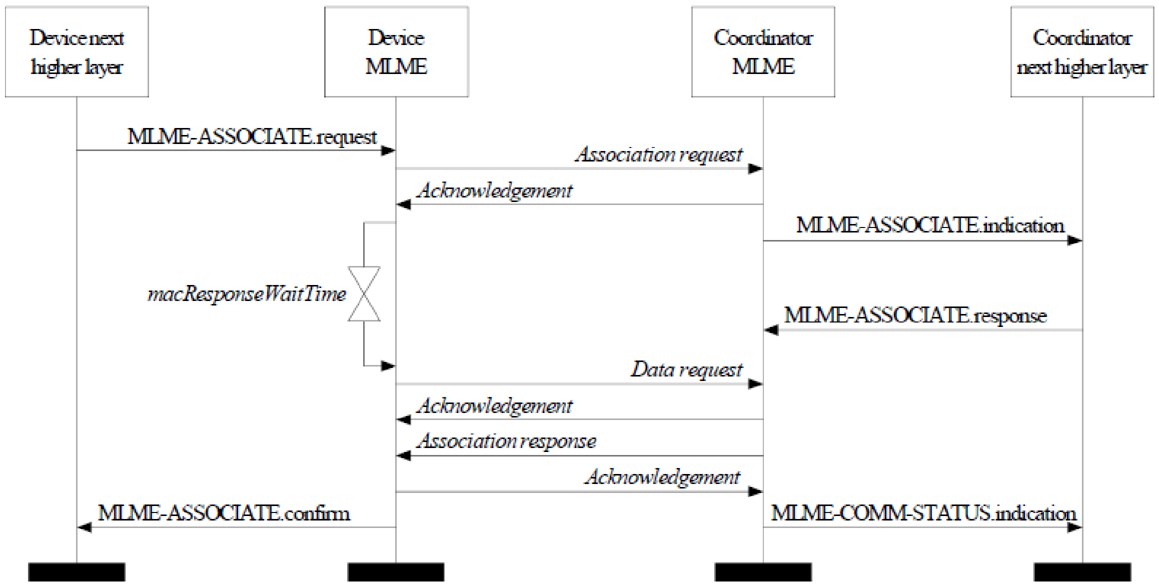


Figure 5 - Sequence diagram of the association primitives

Association primitives

Figure 5 is the sequence diagram showing the message exchange involved in an association.

The MLME-ASSOCIATE.request primitive is generated by the next higher layer of an unassociated device and issued to its MLME to request an association with a PAN through its coordinator. If the device whishes to associate through a coordinator on a beacon-enabled PAN, the MLME may optionnaly track the beacon of that coordinator prior to issuing this primitive.

The MLME-ASSOCIATE.indication primitive is generated by the MLME of the coordinator and issued to its next higher layer to indicate the reception of an association request command.

The MLME-ASSOCIATE.response primitive is generated by the next higher layer of a coordinator and issued to its MLME in order to response to the MLME-ASSOCIATE.indication primitive.

The MLME-ASSOCIATE.confirm primitive is generated by the initiating MLME and issued to its next higher layer in response to an MLME-ASSOCIATE.request primitive. If the request was successful, the status parameter will indicate a successful association, otherwise, the status parameter indicates an error code.

Table 8, Table 9, Table 10 and Table 7 show which are the parameters implemented in NS2.

MLME-ASSOCIATE.request	
Simulator	**Standard**
✓	LogicalChannel
	ChannelPage
✓	CoordAddrMode
✓	CoordPANId
✓	CoordAddress
✓	CapabilityInformation
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 8 - MLME-ASSOCIATE.request parameters

MLME-DATA.indication	
Simulator	**Standard**
✓	DeviceAddress
✓	CapabilityInformation
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 7 - MLME-DATA.indication parameters

MLME-ASSOCIATE.confirm	
Simulator	**Standard**
✓	AssocShortAddress
✓	status
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 9 - MLME-ASSOCIATE.confirm parameters

MLME-ASSOCIATE.response	
Simulator	**Standard**
✓	DeviceAddress
✓	AssocShortAddress
✓	status
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 10 - MLME-ASSOCIATE.response parameters

Differences in the behavior

There are no differences in the behavior.

Disassociation

Disassociation has been disabled in the simulator.

Beacon notification

The MLME-SAP beacon notification primitive defines how a device may be notified when a beacon is received during normal operating conditions.

Beacon notification primitive

The MLME-BEACON-NOTIFY.indication primitive is used to send parameters contained within a beacon frame received by the MAC sublayer to the next higher layer. The primitive also sends a measure of the LQI and the time the beacon was received.

Table 11 displays the parameters of the primitive and states if the parameters are also in NS2 or not.

MLME-BEACON-NOTIFY.indication	
Simulator	**Standard**
✓	BSN
✓	PANDescriptor
✓	PendAddrSpec
✓	AddrList
✓	sduLength
✓	sdu

Table 11 - MLME-BEACON-NOTIFY.indication parameters

Differences

There is no difference in the behavior of the beacon notification primitive.

Reading PIB attributes

The MLME-SAP get primitives define how to read values from the PIB.

Primitives

The MLME-GET.request primitive is generated by the next higher layer and issued to its MLME to obtain information from the PIB.

The MLME-GET.confirm primitive is generated by the MLME and issued to its next higher layer in response to a MLME-GET.request primitive. This primitive returns a status of either SUCCESS, indicating that the request to read PIB attribute was successful or an error code of UNSUPPORTED_ATTRIBUTE.

Table 12 and Table 13 mention the parameters of the primitives present in the simulator.

MLME-GET.request	
Simulator	**Standard**
✓	PIBAttribute
	PIBAttributeIndex

Table 12 - MLME-GET.request parameters

MLME-GET.confirm	
Simulator	**Standard**
✓	status
✓	PIBAttribute
	PIBAttributeIndex
✓	PIBAttributeVAlue

Table 13 - MLME-GET.confirm parameters

Differences in the behavior

On receipt of the MLME-GET.request primitive, the MLME checks to see if the PIB attribute is a MAC PIB attribute or a PHY PIB attribute. [24]
If the requested attribute is a PHY PIB attribute, the request is passed to the PHY by issuing the PLME-GET.request primitive. [24]

In ns2, the MLME does not perform the check. If it is a PHY PIB attribute, the MLME will issue a MLME-GET.confirm with a UNSUPPORTED_ATTRIBUTE.

GTS management

The MLME-SAP GTS management primitives define how GTSs are requested and maintained.

GTS is not implemented in the simulator.

Orphan notification

The MLME-SAP orphan notification primitives define how a coordinator can issue a notification of an orphaned device.

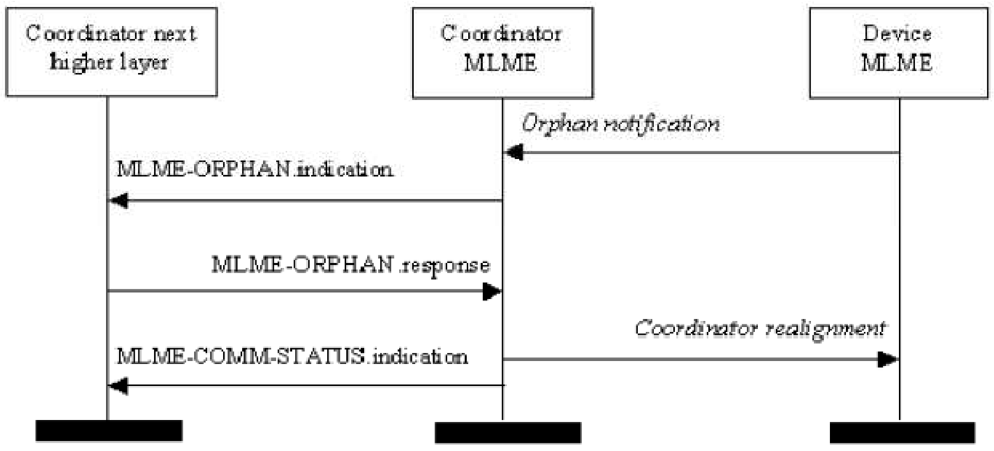


Figure 6 - Sequence diagram of orphan primitives

Orphan notification primitives

Figure 6 shows the exchange of primitives while resolving a case of orphaning.

The MLME-ORPHAN.indication primitive is generated by the MLME of a coordinator and issued to its next higher layer on receipt of an orphan notification command.

The MLME-ORPHAN.response primitive is generated by the next higher layer and issued to its MLME when it reaches a decision about whenever the orphaned device indicated in the MLME-ORPHAN.indication primitive is associated.

Table 14 and Table 15 display the primitives parameters.

MLME-ORPHAN.indication	
Simulator	**Standard**
✓	OrphanAddress
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 14 - MLME-ORPHAN.indication parameters

MLME-ORPHAN.response	
Simulator	**Standard**
✓	OrphanAddress
✓	ShortAddress
✓	AssociateMember
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 15 - MLME-ORPHAN.response parameters

Differences in the behavior

There are no differences in the behavior.

Resetting the MAC sublayer

MLME-SAP reset primitives specify how to reset the MAC sublayer to its default values.

Reset primitives

The MLME-RESET.request primitive is generated by the next higher layer and issued to the MLME to request a reset of the MAC sublayer to its initial conditions.

The MLME-RESET.confirm primitive is generated by the MLME and issued to its next higher layer in response to a MLME-RESET.request primitive and following the receipt of the PLME-SET-TRX-STATE.confirm primitive.

Table 16 and Table 17 mention the primitives parameters.

MLME-RESET.request	
Simulator	Standard
✓	setDefaultPIB

Table 16 - MLME-RESET.request parameter

MLME-RESET.confirm	
Simulator	Standard
✓	status

Table 17 - MLME-RESET.confirm paramter

Differences in the behavior

On receipt of the MLME-RESET.request primitive, the MLME issues the PLME-SET-TRX-STATE.request primitive with a state of FORCE_TRX_OFF. [25]

In ns2, the PLME-SET-STATE.request is issued with a state of TRX_OFF and not FORCE_TRX_FORCE.

On receipt of the MLME-RESET.confirm primitive, the next higher layer is notified of its request to reset the MAC sublayer. This primitive returns a status of SUCCESS indicating that the request to reset the MAC sublayer was successful. [25]

In ns2, there is another status: DISABLE_TRX_FAILURE which is used when the deactivation of TRX failed.

The MLME-RESET.request primitive is issued prior to the use of the MLME-START.request or the MLME-ASSOCIATE.request primitives. [25]

In ns2, the RESET primitive is not issued prior to the use of the START or ASSOCIATE requests.

Specify the receiver enable time

MLME-SAP receiver state primitives define how a device can enable or disable the receiver at a given time.

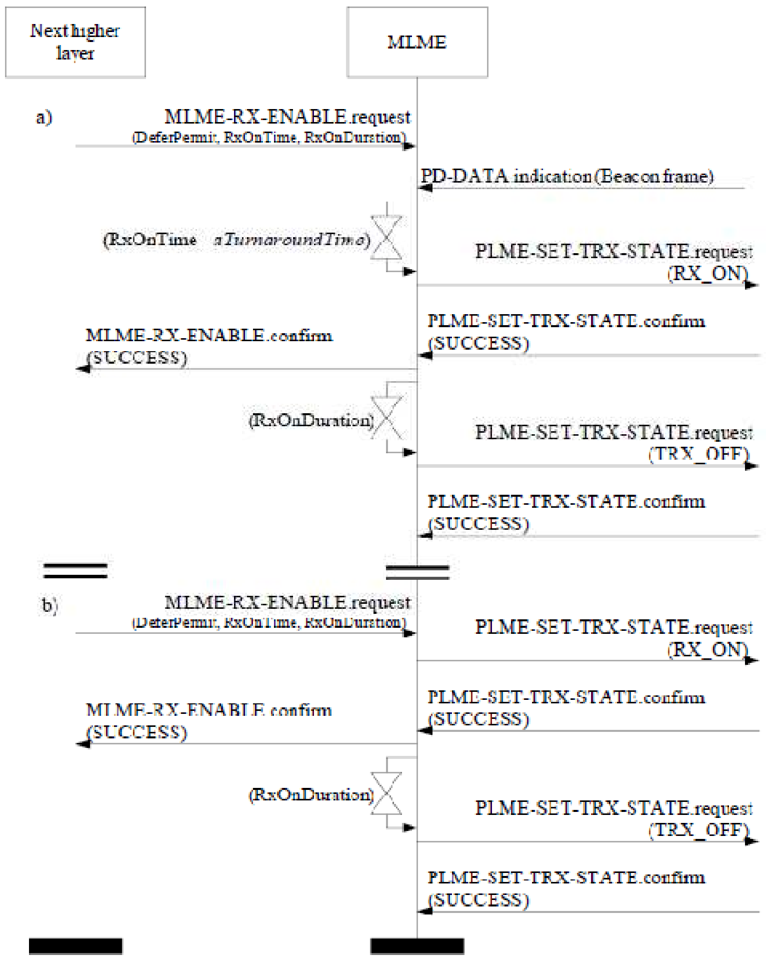

Next higher layer
MLME
a)
MLME-RX-ENABLE.request
(DeferPermit, RxOnTime, RxOnDuration)
PD-DATA.indication(Beacon frame)
(RxOnTime aTurnaroundTime)
PLME-SET-TRX-STATE.request
(RX_ON)
MLME-RX-ENABLE.confirm
(SUCCESS)
PLME-SET-TRX-STATE.confirm
(SUCCESS)
(RxOnDuration)
PLME-SET-TRX-STATE.request
(TRX_OFF)
PLME-SET-TRX-STATE.confirm
(SUCCESS)
b)
MLME-RX-ENABLE.request
(DeferPermit, RxOnTime, RxOnDuration)
PLME-SET-TRX-STATE.request
(RX_ON)
MLME-RX-ENABLE.confirm
(SUCCESS)
PLME-SET-TRX-STATE.confirm
(SUCCESS)
(RxOnDuration)
PLME-SET-TRX-STATE.request
(TRX_OFF)
PLME-SET-TRX-STATE.confirm
(SUCCESS)

Figure 7 - Sequence diagram of MLME-RX primitives

Receiver enable primitives

Figure 7 shows the sequence diagram of the receiver enable primitives.

The MLME-RX-ENABLE.request primitive is generated by the next higher layer and issued to the MLME to enable the receiver for a fixed duration, at a time relative to the start of the currentor next superframe on a beacon-enabled PAN or immediately on a nonBeacon-enabled PAN. This primitive may also be generated to cancel a previously generated request to enable the receiver. The receiver is enabled or disabled exactly once per primitive request.

The MLME-RX-ENABLE.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-RX-ENABLE.request primitive.

Table 18 and Table 19 mention the parameters of the primitives present in the simulator.

MLME-RX-ENABLE.request	
Simulator	**Standard**
✓	DeferPermit
✓	RxOnTime
✓	RxOnDuration

Table 19 - MLME-RX-ENABLE.request parameters

MLME-RX-ENABLE.confirm	
Simulator	**Standard**
✓	status

Table 18 - MLME-RX-ENABLE.confirm parameter

Differences in behavior

If (RxOnTime + RxOnDuration) is not less than the beacon interval, the MLME issues the MLME-RX-ENABLE.confirm primitive with a status of ON_TIME_TOO_LONG. [26]

In ns2, the status used in that case is INVALID_PARAMETERS.

If the MLME cannot enable the receiver in the current superframe and is not permitted to defer the receive operation until the next superframe, the MLME issues the MLME-RX-ENABLE.confirm primitive with a status of PAST_TIME. [26]

In ns2, the status used is OUT_OF_CAP which does not appear are the available status for this primitive.

Channel scanning

MLME-SAP scan primitives define how a device can determine the energy usage or the presence or absence of PANs in a communication channel.

Scan primitives

The MLME-SCAN.request primitive is generated by the next higher layer and issued to its MLME to initiate a channel scan to search for activity within the POS of the device. This primitive can be used to perform an ED scan to determine channel usage, an active or passive scan to locate beacon frames containing any PAN identifier, or an orphan scan to locate a PAN to which the device is currently associated.

The MLME-SCAN.confirm primitive is generated by the MLME and issued to its next higher layer whenbthe channel scan initiated with the MLME-SCAN.request primitive has completed.

Table 20 and Table 21 show the parameters of the primitives.

MLME-SCAN.request	
Simulator	**Standard**
✓	ScanTypel
✓	ScanChannels
✓	ScanDuration
	ChannelPage
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 20 - MLME-SCAN.request parameters

MLME-SCAN.confirm	
Simulator	**Standard**
✓	status
✓	ScanType
	ChannelPage
✓	UnscannedChannels
✓	ResultListSize
✓	EnergyDetectList
✓	PANDescriptorList

Table 21 - MLME-SCAN.confirm parameters

Differences in behavior

If the MLME receives the MLME-SCAN.request primitive while performing a previously initiated scan operation, it issues the MLME-SCAN.confirm primitive with a status of SCAN_IN_PROGRESS. [27]

In ns2, it is checked but no primitive is sent in that case.

If the scan is successful and macAutoRequest is set to FALSE, the primitive results will contain a null set of PAN descriptor values; each PAN descriptor value will be sent individually to the next higher layer using separate MLME-BEACON-NOTIFY primitives. [27]

In ns2, the MLME-BEACON-NOTIFY primitive is not used in this case.

Communication status

The MLME-SAP communication status primitive defines how the MLME communicates to the next higher layer about transmission status when the transmission was instigated by a response primitive and about security errors on incoming packets.

Primitive

Table 22 shows the parameters of the primitive.

MCPS-COMM-STATUS.indication	
Simulator	**Standard**
✓	PANId
✓	SrcAddrMode
✓	SrcAddr
✓	DstAddrMode
✓	DstAddr
✓	status
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 22 - MCPS-COMM-STATUS.indication parameters

Differences in behavior

There are no differences in the behavior.

Writing PIB attributes

MLME-SAP set primitives define how PIB attributes may be written.

Set primitives

The MLME-SET.request primitive is generated by the next higher layer and issued to its MLME to write the indicated PIB attribute.

The MLME-SET.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-SET.request primitive. The MLME-SET.confirm primitive returns a status of either SUCCESS, indicating that the requested value was written to the indicated PIB attribute, or the appropriate error code.

Table 23 and Table 24 mention the parameters of the primitives which are implemented in NS2.

MLME-SET.request	
Simulator	**Standard**
✓	PIBAttribute
	PIBAttributeIndex
✓	PIBAttributeValue

Table 23 - MLME-SET.request parameters

MLME-SET.confirm	
Simulator	**Standard**
✓	status
✓	PIBAttribute
	PIBAttributeIndex

Table 24 - MLME-SET.confirm parameters

Differences in behavior

Once again, ns2 does not check if the attribute is part of MAC PPIB or PHY PIB.

Updating the superframe configuration

MLME-SAP start primitives define how an FFD can request to start using a new superframe configuration in order to initiate a PAN, begin transmitting beacons on an already existing PAN, thus facilitating device discovery or to stop transmitting beacons.

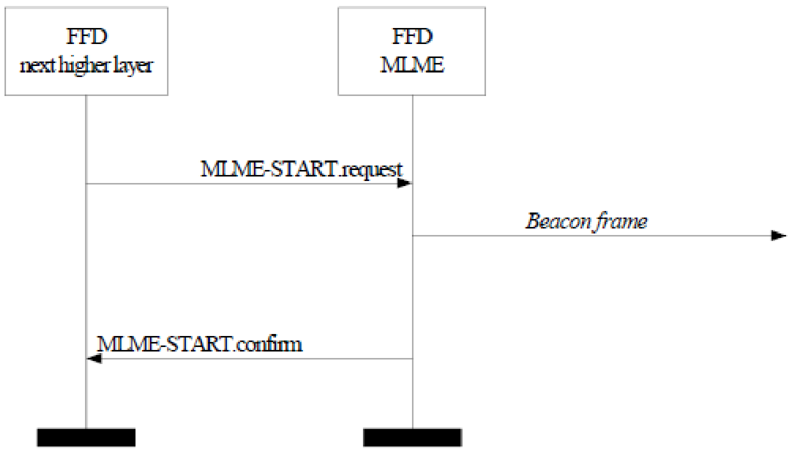


Figure 8 - Sequence diagram for the superframe configuration

Start primitives

Figure 8 shows the sequence diagram for the superframe configuration.

The MLME-START.request primitive is generated by the next higher layer and issued to its MLME to request that a device start using a new superframe configuration.

The MLME-START.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-START.request primitive. The MLME-START.confirm primitive returns a status of either SUCCESS, indicating that the MAC sublayer has started using the new superframe configuration, or the appropriate error code.

Table 25 and Table 26 show the parameters of the primitives.

MLME-START.request	
Simulator	**Standard**
✓	PANId
✓	LogicalChannel
	ChannelPage
	StartTime
✓	BeaconOrder
✓	SuperframeOrder
✓	PANCoordinator
✓	BatteryLifeExtension
✓	CoordRealignment
	CoordRealignSecurityLevel
	CoordRealignKeyIdMode
	CoordRealignKeySource
	CoordRealignKeyIndex
	BeaconSecurityLevel
	BeaconKeyIdMode
	BeaconKeySource
	BeaconKeyIndex

MLME-START.confirm	
Simulator	**Standard**
✓	status

Table 25 - MLME-START.confirm parameter

Table 26 - MLME-START.request parameters

Differences in behavior

If the BeaconOrder parameter is less than 15, the MLME sets macBattLifeExt to the value of the BatteryLifeExtension parameter. [28]

In ns2, macBattLifeExt is set anyway.

If the BeaconOrder parameter is less than 15, the MLME examines the StartTime parameter to determine the time to begin transmitting beacons; the time is defined in symbols and is rounded to a backoff slot boundary. [28]

Ns2 does not use the startTime parameter.

Synchronizing with a coordinator

MLME-SAP synchronization primitives define how synchronization with a coordinator may be achieved and how a loss of synchronization is communicated to the next layer.

Those primitives are used in the case of beacon-enabled network.

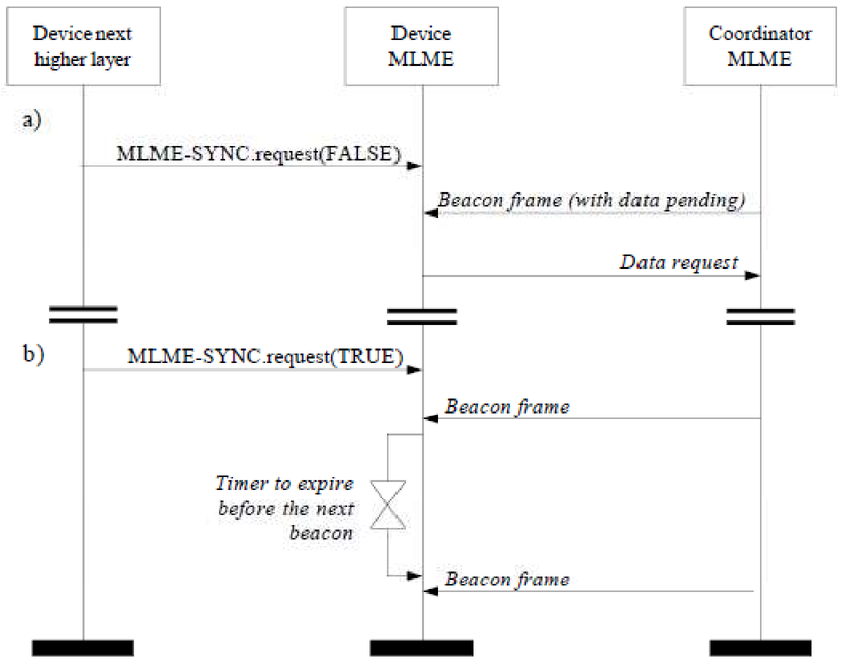


Figure 9 - Sequence diagram for the synchronization primitives

Synchronization primitives

Figure 9 displays the sequence diagram of the synchronization with a coordinator.

The MLME-SYNC.request primitive is generated by the next higher layer of a device on a beacon-enabled PAN and issued to its MLME to synchronize with the coordinator.

The MLME-SYNC-LOSS.indication primitive is generated by the MLME of a device and issued to its next higher layer in the event of a loss of synchronization with the coordinator. It is also generated by the MLME of the PAN coordinator and issued to its next higher layer in the event of a PAN ID conflict.

MLME-SYNC.request	
Simulator	**Standard**
✓	LogicalChannel
	ChannelPage
✓	TrackBeacon

Table 27 - MLME-SYNC.request parameters

MLME-SYNC-LOSS.indication	
Simulator	**Standard**
✓	LossReason
	PANId
	LogicalChannel
	ChannelPage
	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 28 - MLME-SYNC-LOSS.indication

Differences in behavior

One test is performed in ns2 and which is not specified in the standard. The MLME checks if the logical channel is supported. If it is not the case, a MLME-SYNC-LOSS.indication is issued with a UNDIFINED status.

If the PAN coordinator receives a PAN ID conflict notification command, the MLME will issue this primitive with the LossReason parameter set to PAN_ID_CONFLICT. [29]

In NS2, the PAN ID conflict command is not implemented.

If a device has received the coordinator realignment command from the coordinator through which it is associated and the MLME was not carrying out an orphan scan, the MLME will issue this primitive with the LossReason parameter set to REALIGNMENT. [29]

In NS2, this test is not performed while receiving a coordinator realignment command.

Requesting data from a coordinator

MLME-SAP polling primitives define how to request data from a coordinator.

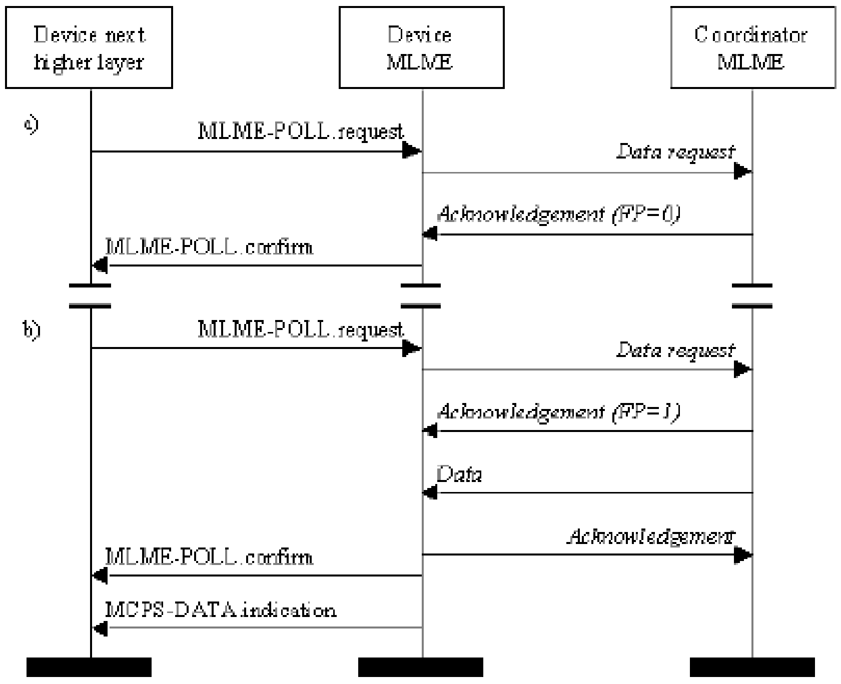


Figure 10 - Sequence diagram for requesting data

Polling primitives

Figure 10 presents the sequence diagram for requesting data to the coordinator.

The MLME-POLL.request primitive is generated by the next higher layer and issued to its MLME when data are to be requested from a coordinator.

The MLME-POLL.confirm primitive is generated by the MLME and issued to its next higher layer in response to an MLME-POLL.request primitive. If the request was successful, the status parameter will be equal to SUCCESS, indicating a successful poll for data. Otherwise, the status parameter indicates the appropriate error code.

MLME-POLL.request	
Simulator	**Standard**
✓	CoordAddrMode
✓	CoordPANId
✓	CoordAddress
✓	SecurityLevel
	KeyIdMode
	KeySource
	KeyIndex

Table 30 - MLME-POLL.request parameters

MLME-POLL.confirm	
Simulator	**Standard**
✓	status

Table 29 - MLME-POLL.confirm parameter

Differences in behavior

There are no differences in the behavior.

3.5. Topologies

One of the topologies which are used in the research field is shown in Figure 11**Erreur ! Source du renvoi introuvable.**.

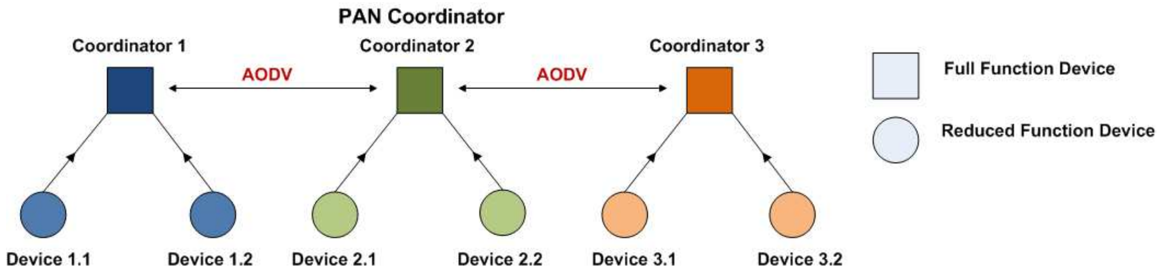


Figure 11 - Topology used in research field

The topology is formed by Devices communicating with Full Function Devices and those Full Function Devices communicating between each other using AODV.

There are two ways to implement that topology in ns2. One is using three different PANs, the other one is to use only one PAN and one of the FFD devices being the PAN coordinator.

3.5.3. With one PAN

One of the FFD has to be the PAN Coordinator. Its functionalities won't be different from the other FFDs, it is only a title. It can be seen as the first device which communicated on the channel. Of course, the other two FFDs will be associated with the PAN coordinator.

The devices connected to the FFDs are RFD, that is, association is not permitted with those nodes.

Figure 12 shows a screenshot from nam, the ns2 interface permitting to watch the simulation events. We can see node 0 which is marked as PAN Coord. The other two FFDs are associated with the PAN Coord and the end devices are associated with their respective coordinator.

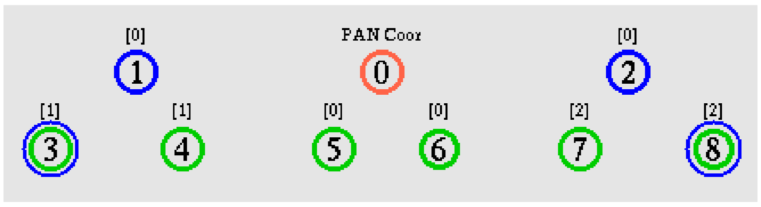


Figure 12 - Nam screenshot of the topology using one PAN

3.5.4. With three PANs

Figure 13 shows the topology with three different PANs, one coordinator and two devices in each PAN.

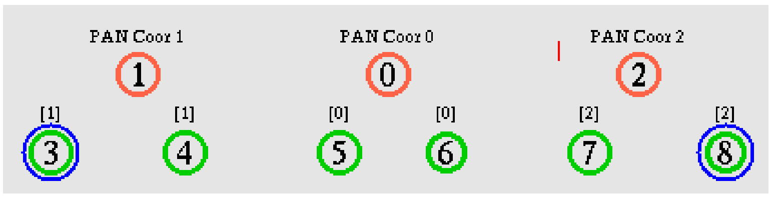


Figure 13 - Nam screenshot of the topology using three PANs

3.5.5. TCL script

In ns2, the topology is specified in a tcl script.

Each node can have a different configuration in terms of power, routing protocol etc. For the differentiation between FFD and RFD, it is done on starting the nodes.

`"$node_(0) sscs startPANCoord"` will be used to start the PAN coordinator. We can also specify here if beacons are enabled. The prototype is:

```
$node_(n) startPANCoord <txBeacon = 1> <beaconOrder = 3>
<SuperframeOrder = 3>
```

`"$node_(1) sscs startDevice 1 1"` will be used to start the other FFDs. They are devices for which association is permitted. They can also transmit beacon. The prototype is:

```
$ node_(n) startDevice <isFFD = 1> <assoPermit = 1> <txBeacon =
0> <beaconOrder = 3> <SuperframeOrder = 3>
```

`"$node_(2) sscs startDevice 0"` will be used to start RFDs, association is not permitted.

The superframe structure, that is the beacon order, can be changed with the following command:

`$ns_ at 9.0 "$node_(5) sscs startBeacon 4 4"` This will change BO and SO value to 4 at the time 9s.

3.6. CBR traffic

CBR stands for Constant Bit Rate. It is very practical to measure performances of 802.15.4 protocol due to its constant bit rate.

In ns2, CBR has to be implemented using a udp agent. Below are the steps to follow to start a CBR traffic.

`set udp_($src) [new Agent/UDP]`	We create a udp agent which will be the source
`eval $ns_ attach-agent \$node_($src) \$udp_($src)`	The source node is attached to the udp agent
`set null_($dst) [new Agent/Null]`	The destination agent is a sink
`eval $ns_ attach-agent \$node_($dst) \$null_($dst)`	The sink is attached to the destination node

`set cbr_($src) [new Application/Traffic/CBR]`	The CBR application is created
`eval \$cbr_($src) attach-agent \$udp_($src)`	The CBR application is attached to the source node
`eval $ns_ connect \$udp_($src) \$null_($dst)`	The udp agent and the sink are connected

Despite the use of the udp agent, when we specify the cbr payload length, the MAC payload length is actually of the length of the cbr payload; udp header is not present. But it is to be understood that the udp header does exist and is used by the application. The udp header is not present in the way that the size of the packet does not take into account the length of udp header but this will only influence the transmission and reception time.

A UDP agent accepts data in variable size chunks from an application and segments the data if needed (more than 1000 bytes). UDP packets also contain a monotonically increasing sequence number and an RTP timestamp. Although real UDP packets do not contain sequence numbers or timestamps, this sequence number does not incur any simulated overhead and can be useful for tracefile analysis or for simulating UDP-based applications.

When the UDP Agent receives a packet, it tests if an application is attached to it. If it is the case, it passes the packet to the application without processing the packet, the application will process it. In the case of CBR, there is no processing of MAC payload because there is nothing to process. The only function of the UDP agent is to transport data.

3.7. ARP

ARP is not part of the 802.15.4 standard. However, by default it is used in order to trigger the transmission of packets from one node to another. It is triggered by the LL layer, so 802.15.4 MAC layer is not aware of it.

It is possible to disable the use of ARP by adding two lines in arp.cc file at the beginning of the function: `int ARPTable::arpresolve(nsaddr_t dst, Packet *p, LL *ll)` (line 125)

```
mac_->hdr_dst((char*) HDR_MAC(p), dst);
return 0;
```

In my case, a weird behavior occurred. ARP was only disabled when I launched ns from Eclipse and was not disabled when I launched it from a command-line terminal.

3.8. AODV

3.8.3. Generalities

AODV routing protocol is implemented in NS2.

To find the adequate route to a destination, it takes into account only the number of hopes and not the quality of the link. So, it will always choose the shortest path and not necessarily the best one.

The AODV protocol uses IP addresses to send messages, therefore, when using AODV, an IP header is appended to each packet, adding 20 bytes of data.

3.8.4. Problems

The RREQ packets from AODV are sent in broadcast to every node it can reach even if they are not in the same PAN. Therefore, the route which is found does pass through several PAN, RFD devices communicating with RFD devices from other PANs and when the device finally sends its data packet to the destination it follows this path.

In order to resolve this problem, we have to apply AODV only for FFD nodes. However, this raises another problem: by default, CBR packets do not use IP header so the destination address is put in the mac destination address field. The RFD node cannot reach that node because it is not in the same PAN so the transmission of the packet is a failure.

3.9. Transmission/reception of a message

In this part a detail list of what happens in NS2 during transmission and reception of a message is conducted.

3.9.3. Conditions

- Non-beacon enabled network
- Packet size: 1 byte
- Acknowledgment
- Node 0 is the coordinator

- Node 1 sends packets to node 0

3.9.4. General behavior

NS2 handles both side of the communication: the receiver and the sender.

The way of working of NS2 is to use a timer which manages the time in the simulation and all the events that take place. Events can be added at a certain time and when the time has come, the event is handled.

NS2 uses also a system of callback. A function can be called back after some events took place. This enables a function to take care of an entire task. For example, in the case of the transmission of a message, one function: MCPS-DATA.request takes care of the entire process, from the construction of the packet, the transmission of the packet to the waiting for the acknowledgment etc.

3.9.5. Steps of the process

Table 31 lists all the step NS2 passes through during transmission and reception of a message.

Node 1: transmitter	Node 0: receiver
Event: Node 1 receives a packet to send from upper layers • It checks if the node is asleep and wakes it up if it the case. • Set the transmission options: Acked requested for example. • Send primitive: MCPS-DATA.request. At this time, the MCPS-DATA.request takes over and handles all the transmission process for node 1. • Construct the mpdu packet Mcps-data-request: Step 0: • Begin Csma: initialization, randomly choose time to wait, start backoff, schedule backoff time to wait in timer. **Event: Backoff time has elapsed** • Set antenna state to reception: RX_ON • Perform CCA. (Normally, CCA lasts 8 symbols. In NS2, CCA lasts 4 symbols but starts after waiting 4 symbols so the entire process does last 8 symbols.	

• Schedule CCA for the 4th symbol). **Event: 4 symbols have elapsed** • Schedule CCA report after 4 symbols, that is, after 8 symbols in total. • Decrease energy from last energy update and beginning of CCA using P_{idle} power. • Decrease energy spent in CCA and turnaround using P_r receive power. **Event: CCA report** Mcps-data-request: Step1: • Enable transmitter: TX_ON • Turnaround **Event: End of the turnaround** • PD_DATA.request • Construction of PPDU • Compute transmission time • Send packet to radio for transmission • Decrease node energy from last energy update using P_{idle} power • Decrease node energy for transmission using $P_{transmission}$ power. • Send packet • Compute propagation delay	
	Event: Packet detected • Test if the packet is for itself.
Event: Packet sent Mcps-data-request: Step 2: • Acknowledgment is required: enable the receiver (turnaround) • Wait for acknowledgment	
	Event: Packet received • Measure link quality • PD_DATA.indication • Check if the packet has to be dropped • Decrease node energy from last update using P_{idle} power • Decrease node energy for reception using $P_{reception}$ power • Construct ack packet • Stop csma if it is pending • Enable transmitter: TX_ON

	• Check duplication of the packet • Delay from aTurnaround symbols **Event: aTurnAround symbols have elapsed** • Construct PPDU for ack packet • Construct transmission time • Send packet to radio • Decrease node energy from last energy update using P_{idle} power • Decrease node energy for transmission using $P_{transmission}$ power • Compute propagation delay
Event: Packet detected • Test if the packet is for itself	
	Event: packet sent • Change state to IDLE • Compute IFS=SIFS • Set trx state to RX_ON
Event: packet received • Measure link quality • PD_DATA.indication • Check if the packet has to be dropped • Decrease node energy from last update using P_{idle} power • Decrease node energy for reception using $P_{reception}$ power • Check duplication • Check sequence number in the ack Mcps-data-request: setp 3: • Task: send data is a success **Event: IFS has elapsed** • Pass packet to upper layers	

Table 31 - Detailed steps for transmission and reception of a single message

3.10. Energy model

In NS2, it is possible to configure an energy model in order to monitor energy consumption of the nodes. This configuration is done in the tcl script.

The parameters of the energy model are:

- initialEnergy: it is the energy available in each node at the beginning of the simulation in Joules
- rxPower: the receiving power in Watts
- txPower: the transmitting power in Watts
- idlePower: the power used when the node is idle in Watts
- sleepPower: the power used when the node is sleeping in Watts
- sleepTime: a parameter used to enable the sleeping mode in Watts

There are two modes available: the default mode and the sleep mode.

3.10.3. Default mode

In this mode, nodes can’t sleep, when they are doing nothing, they are in idle mode and pass in receiving or transmitting mode whenever they are about to receive or transmit a packet.

Note that because NS2 is a simulator, each node knows perfectly well when the reception of a packet will happen.

For the transmitter, energy is decreased after performing the CCA and after the transmission of a packet.

When a turnaround is effectuated, it consumes energy, however, in NS2, the energy of a turnaround is not considered except in the case of the CCA. In the case of the CCA, the power used for the turnaround is the rxPower.

3.10.4. Sleep mode

The sleep mode is activated by the variable sleepTime. sleepTime has to be lower than the current time, that is, to enable sleep mode, we can give sleepTime the value 0.0.

Sleep mode only serves when in beacon-enabled network and only affect RFD nodes.

In that case, the RFD node will pass in sleep node and wake up when transmitting a packet or just before the reception of a beacon packet. Because the RFD node and the coordinator are synchronized, the node is able to predict the instant of reception of such a frame.

The sleep mode permits to save a significant amount of energy.

4. NS2: analysis of performances

In this part, the performances of NS2 will be analyzed and compared with theoretical results or results found in papers.

4.2. Useful bitrate and delay vs Payload size

The first two parameters which will be studied are the useful bitrate and the delay of transmission for one packet; those two parameters being correlated.

The results will be compared to the ones in [30].

In this paper, they study the impact of the length of the addresses used and the usage or not of acknowledgment on the throughput and the delay in 802.15.4 communication.

4.2.3. Conditions

The simulation will be effectuated with the following conditions and topology:

- Two nodes: one sender and one receiver.
- The devices are within range from each other (no loss)
- Channel is considerate perfect (BER=0)
- No losses due to buffer-overflow
- Non-beacon enabled mode
- The node always has enough packets to send
- Propagation: Two-ray model
- Traffic sent: CBR (constant bitrate)
- 2.4 GHz band

The aspects which will be variable are:

- Presence or not of acknowledgment
- Payload size
- Address size: this is done only in the paper, it will not appear in the ns2 simulation

4.2.4. Theoretical study

The maximum throughput is determined as the number of data bits coming from the upper layer (the network layer). Hence, the throughput we are calculating is the throughput of the MAC layer.

We will do the theoretical analysis of the maximum throughput in the case of the unslotted version of the protocol (no super frames) in the 2.4 GHz band, that is, a bitrate of 250 kb/s.

The maximum throughput is calculated as follows:

$$\boldsymbol{TP} = \frac{\mathbf{8}.\boldsymbol{x}}{\boldsymbol{delay}(\boldsymbol{x})} \qquad (1)$$

- x is the size of the payload
- $delay(x)$ is the delay for one packet containing a payload of x bytes.

The delay for each packet is:

$$delay(x) = T_{BO} + T_{frame}(x) + T_{TA} + T_{ACK} + T_{IFS}(x) \qquad (2)$$

• T_{BO}	Back off period
• $T_{frame}(x)$	Transmission time for a payload of x bytes
• T_{TA}	Turn around time (192 µs)
• T_{ACK}	Transmission time for an ACK
• $T_{IFS}(x)$	IFS time (SIFS=192 µs if MPDU<=18 bytes, else, LIFS=640 µs)

Backoff period

$$T_{BO} = BO_{slots}.T_{BO\ slot} \qquad (3)$$

• BO_{slots}	Number of back off slots
• $T_{BO\ slot}$	Time for a back off slot (320 µs)

The number of back off slots is a random number uniformly in the interval $(0.2^{BE} - 1)$, BE being the back off exponent. Its minimum value is 3. The channel is considered to be perfect so we can assume BE will always have the value 3. Therefore, the number of backoff slots is the mean of the interval $(0.2^{3} - 1)$. µ=3.5.

So, $T_{BO} = 3.5 * T_{BO\ slot} = 1.12ms$

Transmission time of a frame with a payload of x bytes

$$T_{frame}(x) = 8.\frac{L_{PHY} + L_{MAC\_HDR} + L_{address} + x + L_{MAC\_FTR}}{R_{data}} \tag{4}$$

- L_{PHY} Length of the PHY header and the synchronization header in bytes (6 bytes)
- L_{MAC_HDR} Length of the MAC header in bytes (3 bytes)
- $L_{address}$ Length of the MAC address fields
- L_{MAC_FTR} Length of the MAC footer in bytes (2 bytes)
- R_{data} Raw data rate of the band used (250 kb/s)

$L_{address}$ in the ns2 simulation is only of 2 bytes so this is the value which will be used for the computation.

Transmission time for an acknowledgment

The way to compute the transmission time for an acknowledgment is the same as the one for an entire frame except that we have no payload and no address field.

$$T_{ACK}(x) = 8.\frac{L_{PHY} + L_{MAC\_HDR} + L_{MAC\_FTR}}{R_{data}} \tag{5}$$

If no acknowledgment is used, T_{TA} and T_{ACK} are omitted.

4.2.5. Results

Hereafter are the results of the useful bitrate and delay without acknowledgment and with acknowledgment afterward. The results have been computed both from the simulation traces and by the theoretical way.

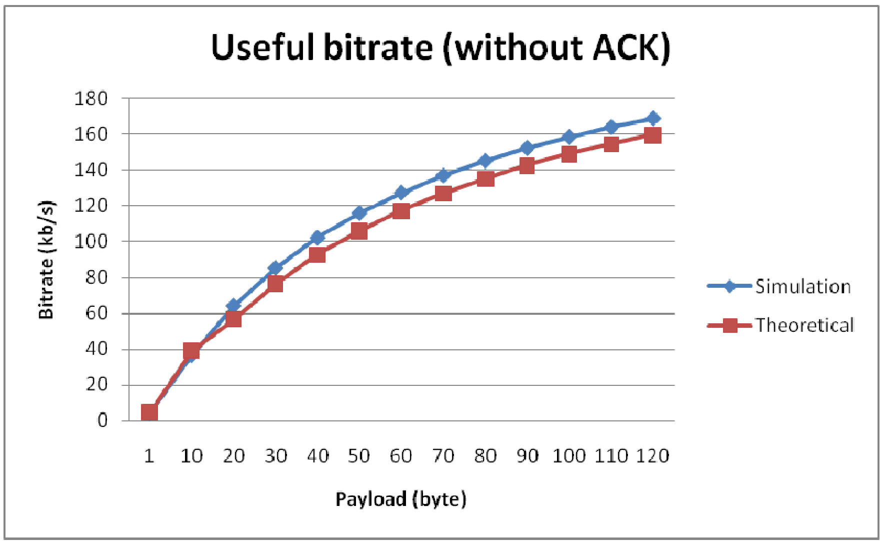
Some graphics follow the result tables.

Without acknowledgment

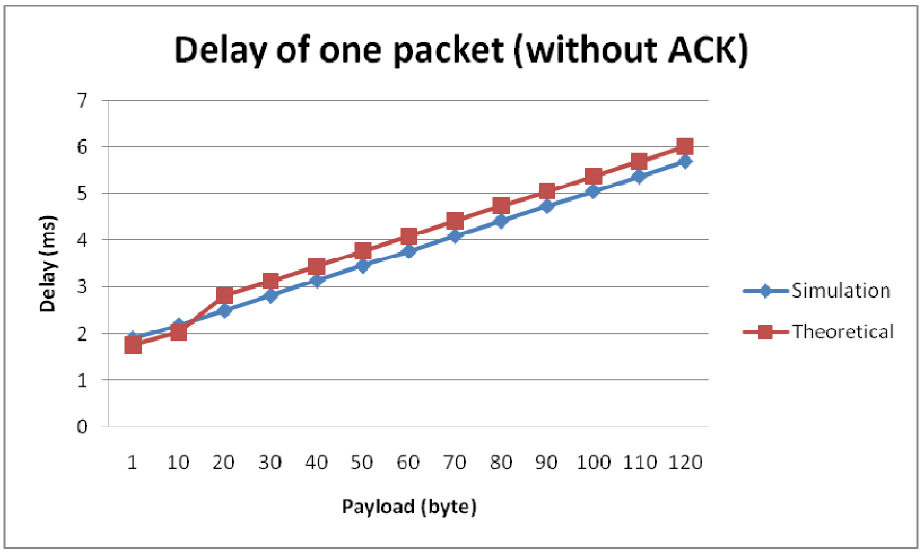
Table 32 shows the numerical results while Graphic 1 and Graphic 2 display the results on a graph.

Useful birate and delay without acknowledgment					
Simulation			Theoretical		
Payload (byte)	Bitrate (kb/s)	Delay (ms)	Payload (byte)	Bitrate (kb/s)	Delay (ms)
1	4,2380	1,8877	1	4,5455	1,760
10	36,7761	2,1753	10	39,0625	2,048
20	64,1072	2,4958	20	56,8182	2,816
30	85,1884	2,8173	30	76,5306	3,136
40	101,9865	3,1377	40	92,5926	3,456
50	115,7281	3,4564	50	105,9322	3,776
60	127,1103	3,7762	60	117,1875	4,096
70	136,7236	4,0959	70	126,8116	4,416
80	144,8822	4,4174	80	135,1351	4,736
90	152,0118	4,7365	90	142,4051	5,056
100	158,1748	5,0577	100	148,8095	5,376
110	163,6141	5,3785	110	154,4944	5,696
120	168,4716	5,6983	120	159,5745	6,016

Table 32 - Useful bitrate and delay without acknowledgment



Graphic 1 - Useful bitrate without acknowledgment



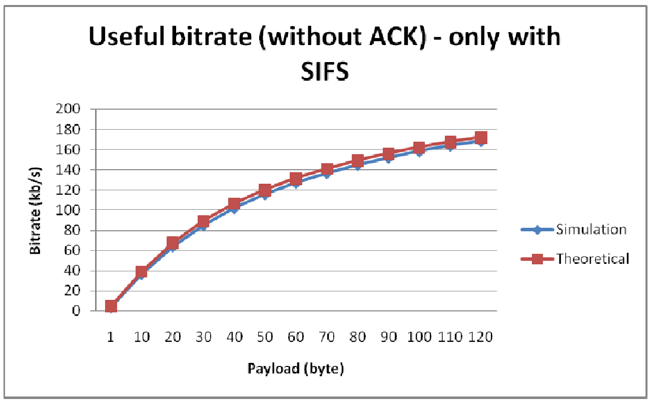
Graphic 2 - Delay of one packet without acknowledgment

Discussion

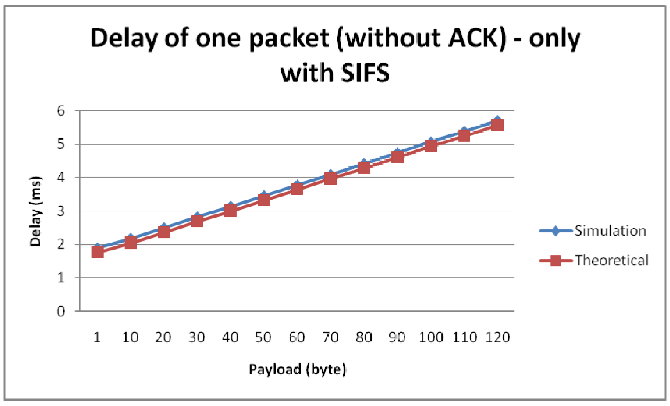
We can see that there are some differences between the theoretical result and the simulation result.

In particular, we can see a transition in the curve of the theoretical graph both for the bitrate and the delay but we can see it more clearly in Graphic 2. This break corresponds to the transition from SIFS to LIFS. However, we cannot see this transition in the simulation curve so we can think that there may be a problem in the simulator concerning the choice of SIFS and LIFS. Though I have found a lot of code lines in the simulator concerning this transition: some tests on the MPDU size are performed to see which interval to choose, there seems to have a flaw.

I tried to change the theoretical curve using the SIFS interval for all payload sizes to see if the two curves would match better. And indeed, we can see in Graphic 3 and Graphic 4 that it is a nearly perfect match even if the bitrate for the theoretical curve is slightly higher.



Graphic 3 - Useful bitrate (without ACK) - using only SIFS interval



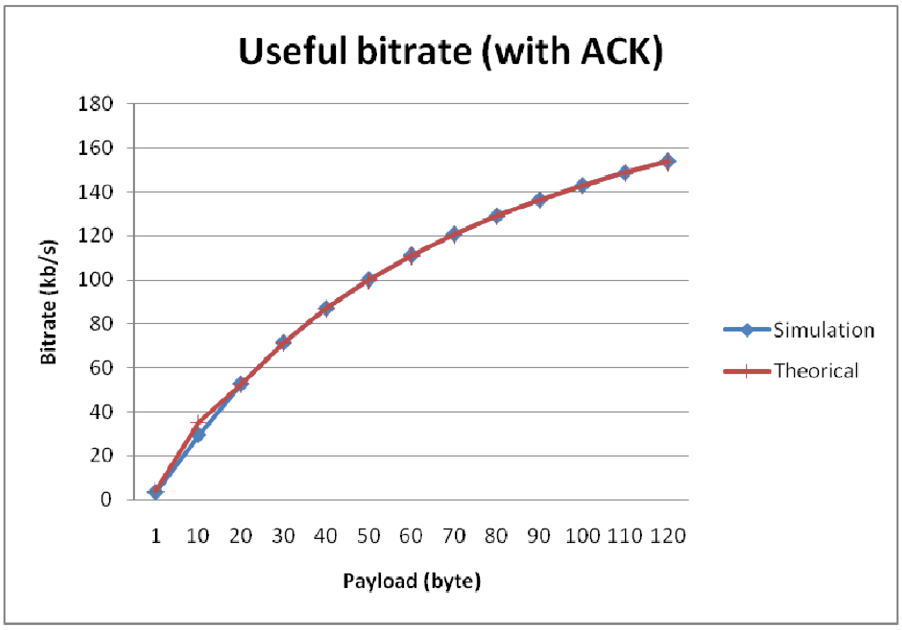
Graphic 4 - Delay of one packet (without ACK) - using only SIFS interval

With acknowledgment

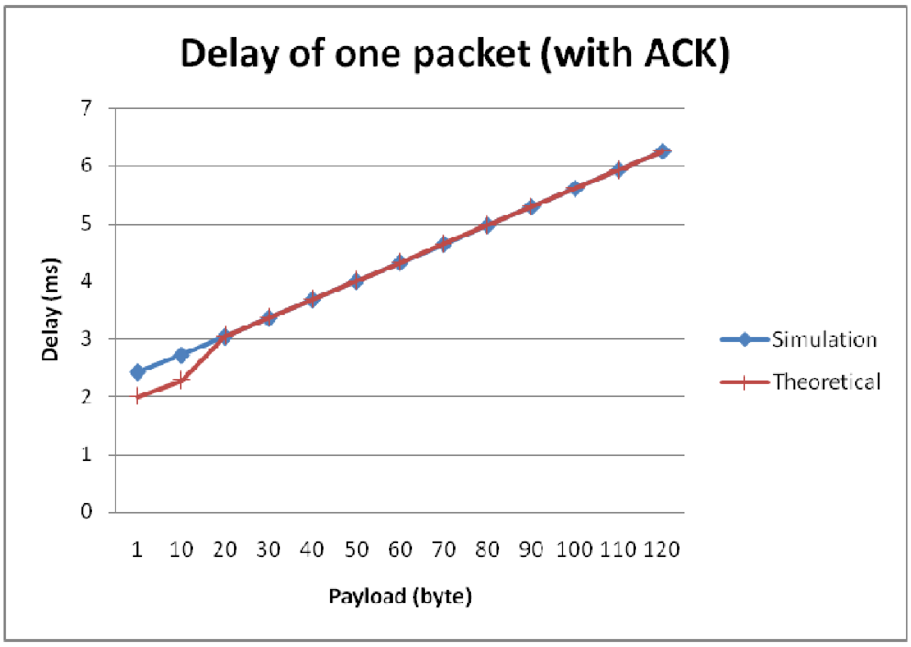
Table 33 shows the numerical results while Graphic 5 and Graphic 6 display the results on a graph.

Useful birate and delay with acknowledgment					
Simulation			Theoretical		
Payload (byte)	Bitrate (kb/s)	Delay (ms)	Payload (byte)	Bitrate (kb/s)	Delay (ms)
1	3,2896	2,4319	1	4,0080	1,996
10	29,4019	2,7209	10	35,0263	2,284
20	52,5930	3,0422	20	52,4246	3,052
30	71,4122	3,3608	30	71,1744	3,372
40	86,9275	3,6812	40	86,6739	3,692
50	99,9971	4,0001	50	99,7009	4,012
60	111,0541	4,3222	60	110,8033	4,332
70	120,6643	4,6410	70	120,3783	4,652
80	128,9800	4,9620	80	128,7208	4,972
90	136,2969	5,2826	90	136,0544	5,292
100	142,7817	5,6030	100	142,5517	5,612
110	148,5873	5,9224	110	148,3479	5,932
120	153,8094	6,2415	120	153,5509	6,252

Table 33 - Useful bitrate and delay with acknowledgment



Graphic 5 - Useful bitrate with acknowledgment

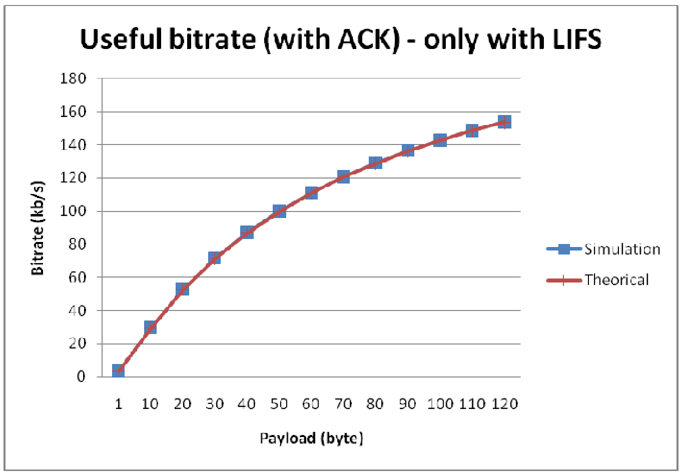


Graphic 6 - Delay of one packet with acknowledgment

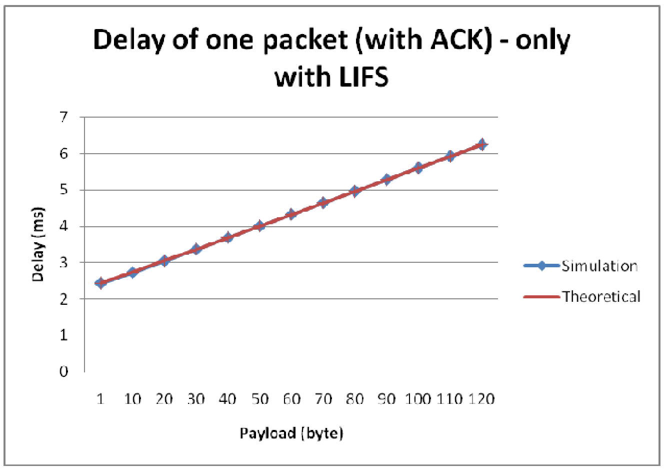
Discussion

Once again, we can see that there is a problem of transition between SIFS and LIFS. This time, it is the second part of the curves that best match. See Graphic 6.

So, I changed the theoretical curve to use always LIFS and Graphic 8 and Graphic 7 show that the result is a perfect match. We can see it in the tables above, for the LIFS part of the curve, the values are almost the same.



Graphic 8 - Useful bitrate (with ACK) - using only LIFS interval



Graphic 7 - Delay of one packet (with ACK) - using only LIFS interval

4.3. Useful bitrate and delivery ratio vs Number of devices

The following two studies refer to results found in [31].

4.3.3. Conditions

The conditions for those two experiments are the same: one coordinator and one to four devices in a star topology. There will be one main transceiver with the coordinator, the other devices being only traffic load generators. See Figure 14. All the results are computed from device 1.

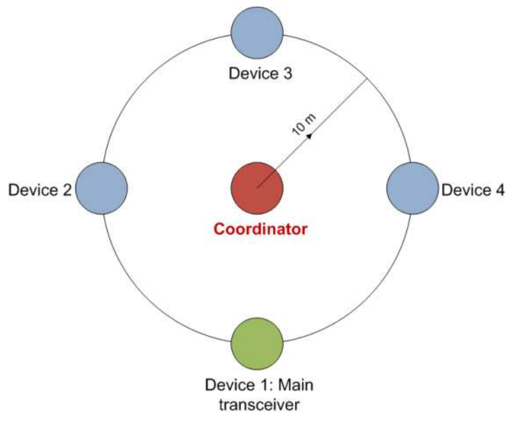


Figure 14 - Star topology with one coordinator, one main transceiver and three load generators

- Every device can hear every other device; there is no problem of hidden terminal.
- The aMaxFrameRetries parameter has been set to 0. This way, there is no retransmission in case of drop packet.
- Non-beacon enabled mode

The results will be compared to ones in [31]. The author realized the experiments using IEEE 802.15.4 development boards.

I have to mention beforehand that it will not be possible to compare expressively the values obtained because the conditions are somewhat dissimilar but it will be possible to compare the tendencies and in certain cases we will be able to reuse the results obtained during the previous simulations.

4.3.4. Varying traffic load per device

Here, the effective data rate and the delivery ratio will be computed for 1, 2, 3 and 4 devices, varying the traffic load per device other than the main transceiver with the values 10kb/s, 50kb/s and 100kb/s.

I have to remind that the value of the bitrate and the delivery ratio are computed according to the simulation traces of device 1, the main transceiver, with the coordinator.

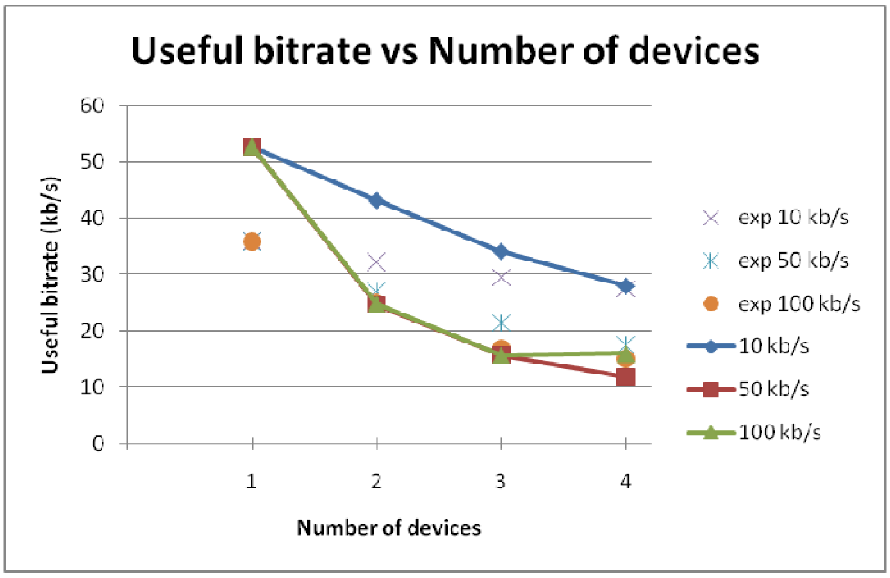
Results

Table 34 shows the numerical results while Graphic 9 and Graphic 10 display the results on a graph.

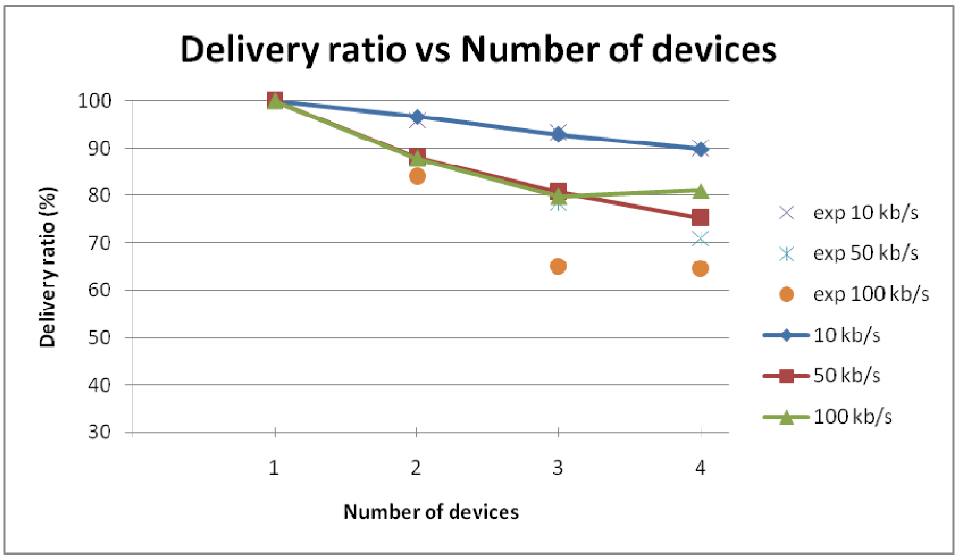
On the two graphics, the values marked with "exp" are the ones retrieved from the experimental results from the reference paper.

Useful birate and delivery ratio (Varying traffic load)								
10 kb/s			50 kb/s			100 kb/s		
Number of devices	Bitrate (kb/s)	Delivery ratio (%)	Number of devices	Bitrate (kb/s)	Delivery ratio (%)	Number of devices	Bitrate (kb/s)	Delivery ratio (%)
1	52,5940	100,0000	1	52,5940	100,0000	1	52,5940	100,0000
2	43,0916	96,6859	2	24,6800	88,1267	2	24,7497	87,8760
3	34,0687	92,9117	3	15,8290	80,7808	3	15,6522	79,8357
4	27,9635	89,7804	4	11,9015	75,2593	4	15,9795	81,1338

Table 34 - Useful bitrate and delivery ratio (Varying traffic load)



Graphic 9 - Useful bitrate vs Number of devices (Varying traffic load)



Graphic 10 - Delivery ratio vs Number of devices (Varying traffic load)

Discussion

<u>Useful bitrate</u>

In Graphic 9, we can see that the tendencies of the curves are similar. We can even observe a similarity in the values for the highest numbers of devices.

In my case, the two curves for 50 kb/s and 100 kb/s are equal which seems logical. Indeed, the maximum bitrate for the load generators is inferior to 50 kb/s in all cases so it would not change anything to have 50kb/s in one hand and 100kb/s in the other hand. The results should be the same.

However we can see that in the case of 4 devices, the two curves have different values, we can also observe this in Graphic 10.

<u>Delivery ratio</u>

In Graphic 10, the two curves are very similar both in tendency and in values. We can see that the curves for 10 kb/s and 50 kb/s seem to have the same values than the ones in the reference paper.

Once again, in my case 50 kb/s and 100 kb/s show the same results and once again, it seems to have some problem for 100 kb/s in the case of 4 devices.

4.3.5. Varying data payload

The simulations here are similar to the previous ones. However, the traffic load for devices 2, 3 and 4 is fixed to 100 kb/s and it is the payload size which varies.

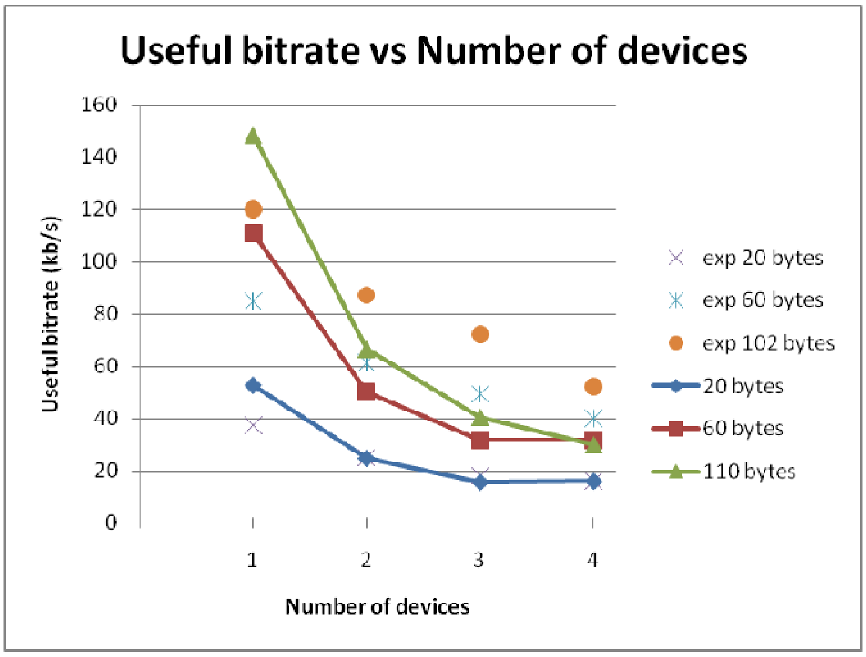
But give attention to the fact that even if the payload size may be the same regarding the reference paper; the size of the packet can be different because in the paper there was no information about the size of the address field.

Once again, the results marked with "exp" are from the reference paper.

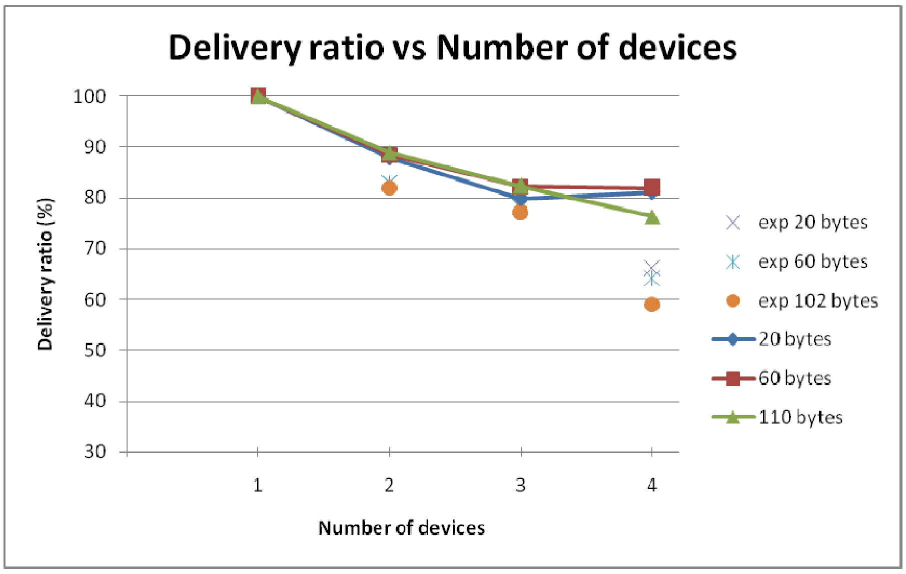
Results

Useful birate and delivery ratio (Varying data payload)								
20 bytes			60 bytes			110 bytes		
Number of devices	Bitrate (kb/s)	Delivery ratio (%)	Number of devices	Bitrate (kb/s)	Delivery ratio (%)	Number of devices	Bitrate (kb/s)	Delivery ratio (%)
1	52,5940	100,0000	1	111,0541	100,0000	1	148,5873	100,0000
2	24,7497	87,8760	2	50,0273	88,5226	2	66,4807	88,9535
3	15,6522	79,8357	3	31,6181	82,1933	3	40,3531	82,2731
4	15,9795	81,1338	4	31,5944	81,9391	4	29,8511	76,3514

Table 35 - Useful bitrate and delivery ratio (varying data payload)



Graphic 11 - Useful bitrate vs Number of devices (Varying payload size)



Graphic 12 - Delivery ratio vs Number of devices (Varying payload size)

Discussion

We can do almost the same remarks as for the previous simulations. In Graphic 11 and Graphic 12, the tendencies of the curves are similar. This is true above all for the delivery ratio. We can see that for the useful bitrate, in the case of the simulation, the curves decrease faster.

We can observe also the same problem for the case of 4 devices.

4.4. Useful bitrate vs Beacon order

For this experiment, we will study the variation of the bitrate with the beacon order. Figure 15 explains the structure of a superframe.

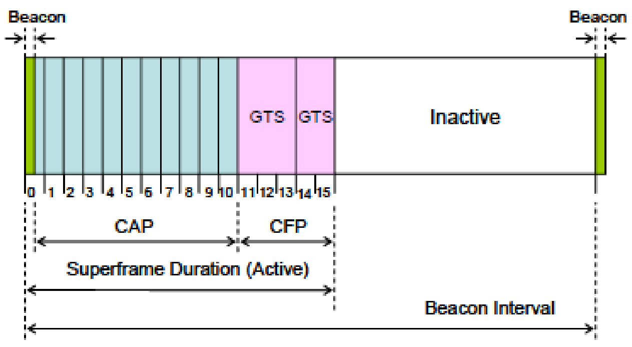


Figure 15 - Superframe structure

The beacon order (BO) and the superframe order (SO) define the structure of the superframe. The superframe is bounded by two beacons. It contains an active part in which transmission can be effectuated and an inactive part, during which the coordinator enters in a sleep mode.

The superframe length is function of the beacon order:

$$BI = aBaseSuperFrameDuration * 2^{BO} \qquad (6)$$

Where BI stands for Beacon Interval.

The active part length depends on the superframe order:

$$SD = aBaseSuperFrame * 2^{SO} \qquad (7)$$

Where SD stands for Superframe Duration.

aBaseSuperFrameDuration is the number of symbols forming a superframe when the superframe order is equal to 0. It is equal to 960 symbols (1 symbol corresponds to 4 bits).

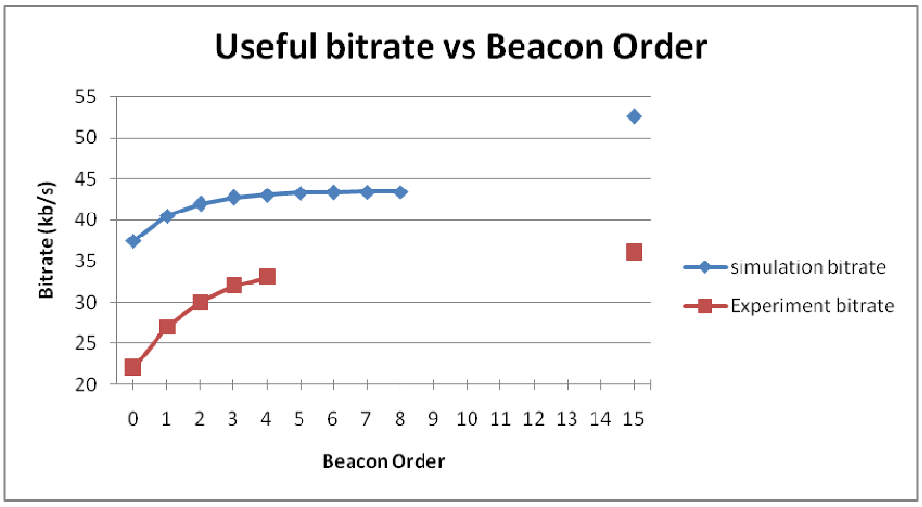
In our case, BO=SO, that is, there is no inactive part. A BO equal to 15 refers to a non-beacon enabled mode.

Result

Table 36 shows numerical results and Graphic 13 displays the results on a graph.

Useful bitrate		
vs Beacon order		
Beacon order	Bitrate (kb/s)	Beacon Interval (s)
0	37,486453	0,01536
1	40,506306	0,03072
2	41,975134	0,06144
3	42,691877	0,12288
4	43,055200	0,24576
5	43,263008	0,49152
6	43,357714	0,98304
7	43,408000	1,96608
8	43,432000	3,93216
9	NO DATA	7,86432
10	NO DATA	15,72864
11	NO DATA	31,45728
12	NO DATA	62,91456
13	NO DATA	125,82912
14	NO DATA	251,65824
15	52,592982	∞

Table 36 - Useful bitrate varying beacon order



Graphic 13 - Useful bitrate vs Beacon order

Discussion

We can see that there are no data for a beacon order superior to 8. Indeed, the association could not manage to succeed because the scanning time in NS2 is inferior to 7 seconds and beacon interval for an order of 9 or superior is higher than 7 seconds.

We can observe that the curve is quite similar to the one in the paper. However, the value for the non-beacon enabled mode (52,6 kb/s) is much higher than the value where the curve seems to stagnate (43,5 kb/s).

In the paper's graphic, the curve increases till the non-beacon enabled mode. However, it seems to be more an extrapolation of the curve than real results.

4.5. Energy model

Ns2 802.15.4 module provides an energy model from which it is possible to monitor energy consumption for each node.

The following study has been realized taken into account results obtained by Carolina Tripp in her master thesis. She made an analysis of the energy consumption depending on the payload size and the presence or not of security.

The same conditions have been recreated in ns2. However, the study will not involve security since security is not implemented in ns2.

The motes used are TelosB. From Carolina work, it has been found that applying a voltage of 3V:

- Transmission mode current: 24 mA

- Reception mode current: 26 mA
- Idle mode current: 4.7 mA

In NS2, we have to put the value of the power for each mode, with $P = U.I$ we obtain:

- Transmission mode power: 72 mW
- Reception mode power: 78 mW
- Idle mode power: 14.1 mW

The simulation has been made transmitting 20 CBR packets with different packet size. The result has then been divided to get the result for 1 packet.

A theoretical study has been made in order to compare the two results.

4.5.3. Theoretical study

To compute the energy consumed for the total transfer of a frame, it has been considered:

- The energy consumed during the transmission or reception of the frame
- The energy consumed during backoff time and CCA (8 symbols) for the transmitter or idle time for the receiver

The general formula used is

$$E = IUd \tag{8}$$

where,

- E is the energy (J)
- I is the current (A)
- d is the delay (s)
- U is the voltage applied (V)

And:

$$E_{trans} = E_{backoff} + E_{idle} + E_{CCA} + E_{transmission} \tag{9}$$

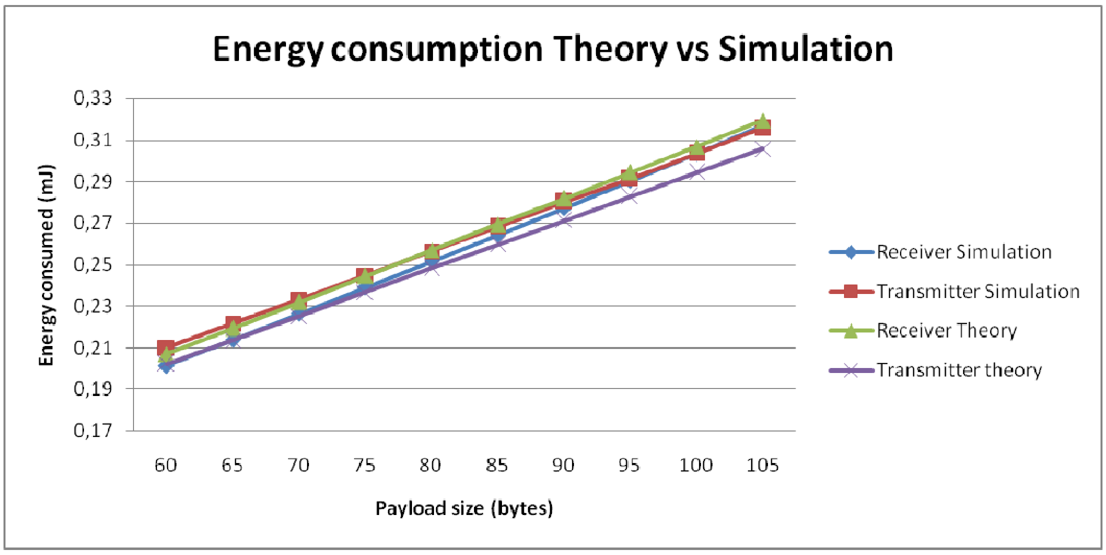
$$E_{recv} = E_{idle} + E_{reception} \tag{10}$$

4.5.4. Results

Table 37 shows numerical results and Graphic 14 displays those results.

Energy consumption for 1 packet					
Simulation			Theoretical		
Payload (byte)	Transmitter energy (mJ)	Receiver energy (mJ)	Payload (byte)	Transmitter energy (mJ)	Receiver energy (mJ)
60	0,210	0,201	60	0,202	0,207
65	0,222	0,214	65	0,214	0,220
70	0,233	0,226	70	0,225	0,232
75	0,245	0,239	75	0,237	0,244
80	0,257	0,252	80	0,248	0,257
85	0,268	0,264	85	0,260	0,269
90	0,280	0,277	90	0,271	0,282
95	0,291	0,290	95	0,283	0,294
100	0,304	0,304	100	0,294	0,307
105	0,316	0,317	105	0,306	0,319

Table 37 - Energy consumption for 1 packet



Graphic 14 - Energy consumption: Theory vs Simulation

Discussion

In Graphic 14, we can clearly see that the result do match, the energy consumption in the simulator is similar to the one found theoretically.

4.5.5. Comparaison sleep mode vs non sleep mode

In sleep mode, the network needs to be in a beacon-enabled mode. It has been tested with Bo and SO equal to 3.

The results of Table 38 are those of the RFD, which is the transmitter energy at the end of the sending of 21 CBR packets. The beginning of the transmission happens at time 30 seconds.

The sleep power is 1.44e-7 Watts.

Energy consumption			
Sleep mode		Non-sleep mode	
Payload (byte)	Transmitter energy (mJ)	Payload (byte)	Transmitter energy (mJ)
60	0,10540	60	0,42772
65	0,10570	65	0,42795
70	0,10585	70	0,42818
75	0,10605	75	0,42841
80	0,10632	80	0,42864
85	0,10654	85	0,42888
90	0,10678	90	0,42912
95	0,10700	95	0,42934
100	0,10724	100	0,42958
105	0,10743	105	0,42983

Table 38 - Energy consumption (sleep mode/non sleep mode)

We can see that in this case, the use of the sleep mode saves about 0,35 mJ of energy. We can see that in beacon-enable mode, the size of the payload does not really impact the consumption of energy.

4.6. Conclusion

Regarding the results obtained above, we can say that ns2 simulator seems to be fairly reliable and that it is a good choice for the simulation of the sensor network embedded on a train.

5. Protocol simulation

In this part, I will first describe the architecture of the network wanted as well as the protocol itself. I will then state the alterations made in order to perform the simulation and the results of that simulation.

5.2. Architecture

The network is deployed on a train. It is formed by one gateway, situated at the head of the train, 7 relays spaced by 25 meters along the train and sensors placed on the train wheels. There are 4 sensors for each relay.

The sensors are in charge of acquiring temperature and vibration information. They do not run any routing protocol and are associated with a relay node.

The relay nodes are relaying data from the sensors to the gateway and vice versa.

The gateway collects all data coming from sensors and can exert some control on the sensors by sending control frames to them.

Figure 16 is a scheme of the network as deployed on the train. The scheme is a side view and therefore shows only two sensors for each relay instead of four.

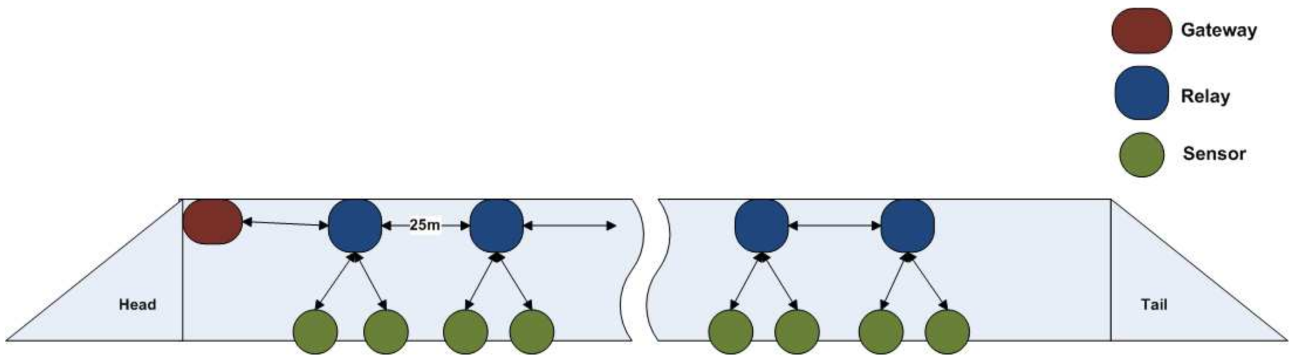


Figure 16 - Network architecture

5.3. Protocol

The network is formed by the repetition of a cycle formed by three steps. First of all, the sensor will send a temperature message to the gateway. Then, the gateway will respond to the sensor and finally, depending on the gateway's response, the sensor will send vibration information or go to sleep.

Sending the temperature message

Each sensor periodically sends a temperature message to the gateway with a period of one minute. The sensors are not started up exactly at the same time so the temperature messages are not sent either at the same time.

To send its message to the gateway, the sensor will send it directly to its relay node. The relay will then transmit the message to the gateway using AODV protocol.

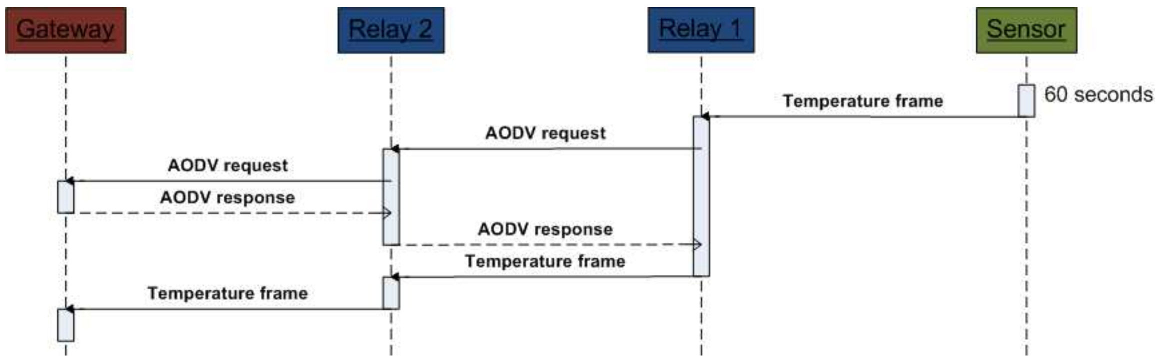


Figure 17 - Sequence diagram: the sensor sends a temperature frame

The gateway response

Once it has received the temperature message from the sensor node, the gateway needs to reply. It will send a control message.

Two cases can be seen:

- The gateway sends a “go to sleep” message.
- The gateway asks for vibration information.

The gateway will ask to the sensors to send vibration information in a round robin fashion. When the gateway receives a temperature message form one node, it will verify if it is this node’s turn to send vibration data. If it is the case, it will ask for the data or else it will tell the node to go to sleep.

When the sensor node goes to sleep, it will remain in that state until the 60 seconds have elapsed and it is then time to send another temperature frame.

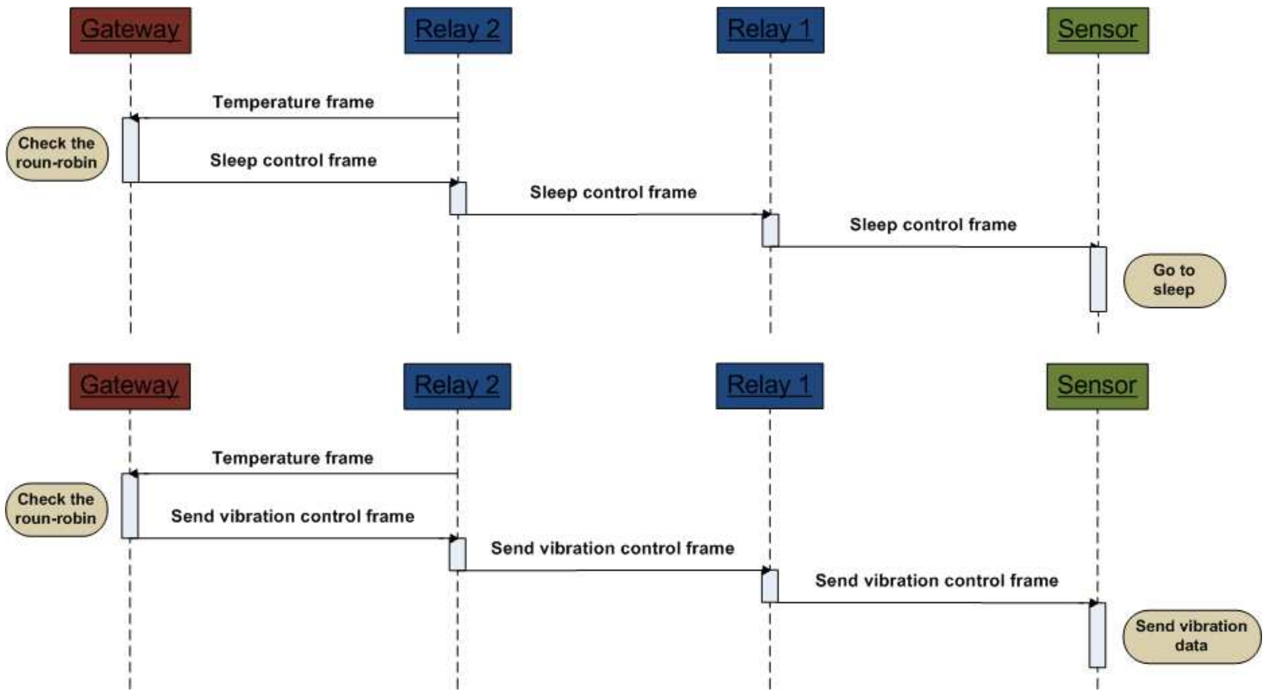


Figure 18 - Sequence diagram: the gateway response

If the temperature frame or the control frame is lost during its transmission, the sensor will not receive any response. If the node does not receive anything during 5 seconds, it will automatically go to sleep.

Sending the vibration message

Once the sensor node receives the “Send vibration control frame”, it will begin the transmission of the vibration data. The transmit window is 10 packets.

The sensor will begin by sending 10 packets.

If everything goes fine and no packet is lost, the gateway will send an ACK message after receiving 5 packets correctly. When receiving the ACK message, the sensor sends 5 more packets. See Figure 19.

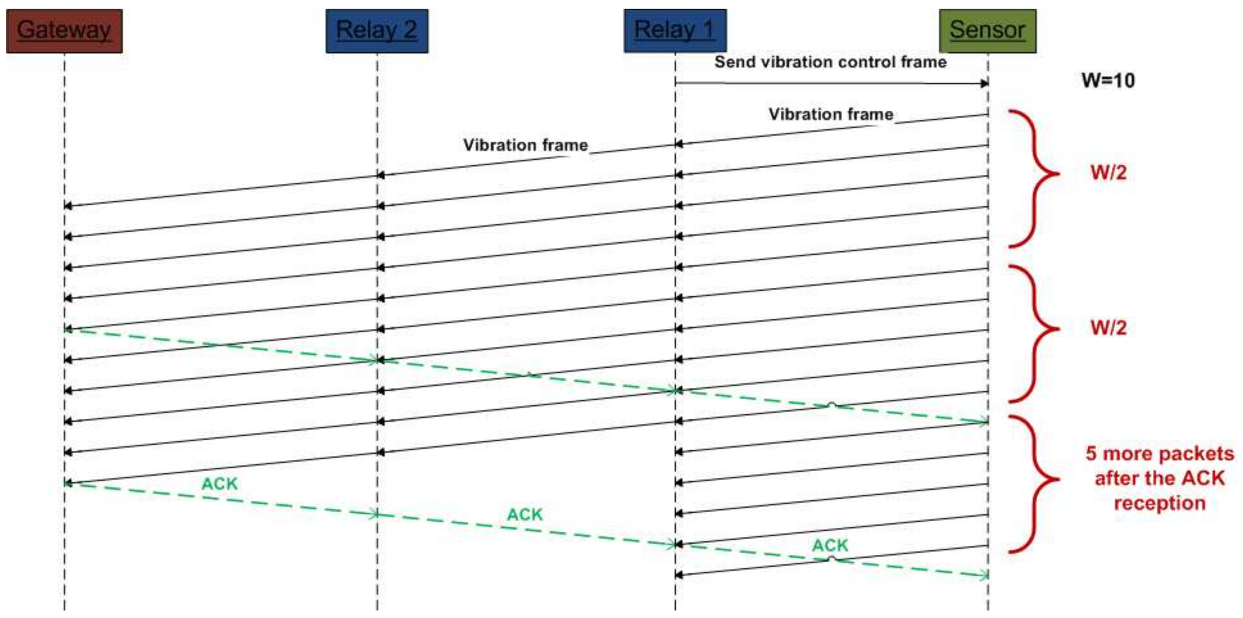


Figure 19 - Sending vibration data with no loss

If a packet is lost, the gateway sends a N-ACK message to the sensor node for the lost message. N-ACK message implies that every vibration messages with a sequence number inferior to the one specified in the N-ACK frame have been received properly. When receiving a N-ACK frame, the sensor first sends back the lost vibration packet. Then, the sensor sends n vibration packets; n being the number of packets properly received that the N-ACK acknowledged. See Figure 20.

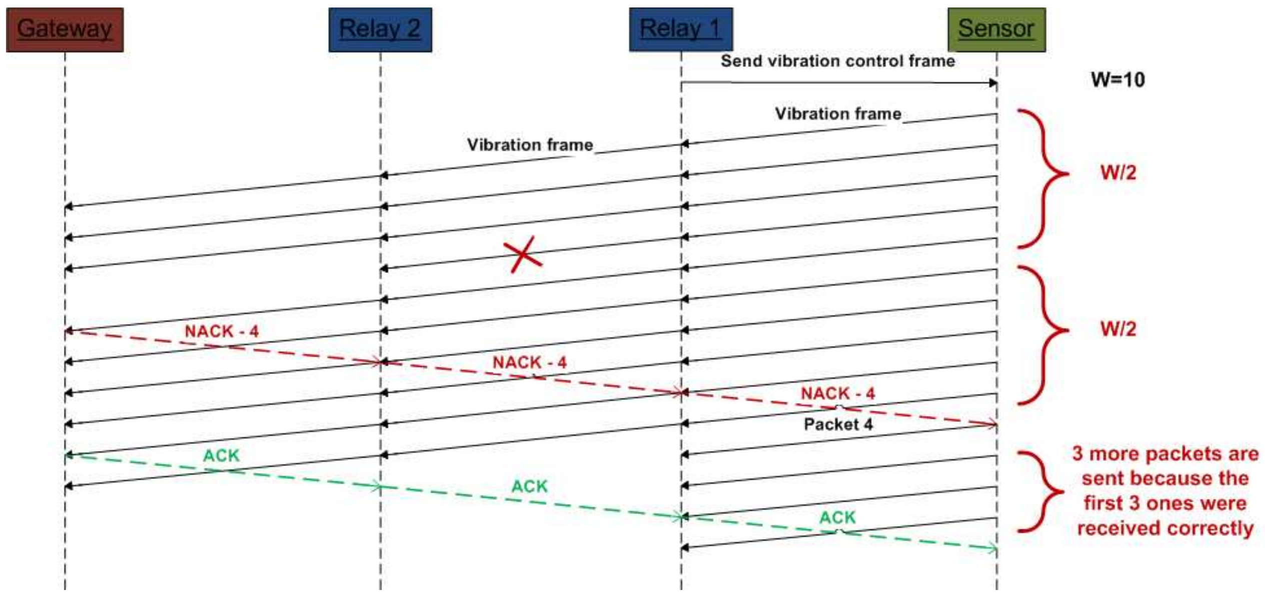


Figure 20 - Sending vibration data with losses

If during a defined time the gateway does not receive any vibration message, it will send a N-ACK message for the last vibration message not received.

Frames details

General frame

Each frame is composed by the following fields:

- MAC header: information of MAC level (used in 802.15.4 frames)
- Serial header: information of Serial protocol level (used in serial protocol frames)
- AM header: information about the type of frame
- AODV header
- Payload field

The MAC header (9 bytes) is composed by:

- Frame control: 2 bytes
- Sequence number: 1 byte
- Destination PAN address: 2 bytes
- Destination address: 2 bytes
- Source address: 2 bytes

The Serial Protocol header (7 bytes) is composed by:

- AM type: 1 byte
- Destination address: 2 bytes
- Link source packet: 2 bytes
- Message length: 1 byte
- Group ID: 1 byte

AODV header for Data frames (9 bytes) is optional and is composed by:

- Single hop sequence number: 1 byte
- Multi hop source address: 2 bytes
- Multi-hop destination address: 2 bytes
- Application field: 1 byte
- Multi-hop sequence number: 1 byte
- TTL field: 1 byte
- Flag field (optional): 1 byte

Payload field: (variable length and format):

- Data temperature
- Data vibration
- Control
- Association

Data temperature frame payload field

The temperature frame has a length of 6 bytes.

- Sensor node source address: 2 bytes
- Sensor relay source address: 2 bytes
- Temperature Data: 2 bytes

Data vibration frame payload field

The vibration frame has a variable length. Its maximum length is of 127 bytes.

- Sensor node source address: 2 bytes
- Sensor relay source address: 2 bytes
- Sequence number: 9 bits
- Data length: 7 bits
- Vibration data: 121 bytes maximum

Control frame payload field

From the gateway to sensor nodes, the frame has a length of 5 bytes

- Sensor node destination address: 2 bytes
- Sensor relay destination address: 2 bytes
- Control type: 1 byte

The control type can be:

- 0x00: Sleep
- 0x01: Send vibration information
- 0x03: Change value of temperature acquisition period

From a sensor to the gateway, the frame has a length of 3 bytes.

- Sensor node source address: 2 bytes
- Control type: 1 byte

The control type is: 0x80: ACK – Action done

5.4. The simulation

To simulate this protocol, I used the ns2 simulator and its 802.15.4 physical and MAC layers. I then programmed three entire application layers, one for each type of sensor: sensor, relay and gateway. I also had to modify the UDP layer to take into account the new application layer.

I gave an address to each sensor depending on its location on the train. The gateway has address 0, then for each relay, the address is the previous one adding ten. The wheel sensors have the address of the relay they relate to adding one, two, three or four. Figure 21 shows the disposition of the nodes and their addresses. Each relay has a reach range of 50 meters, that is two relays in each direction.

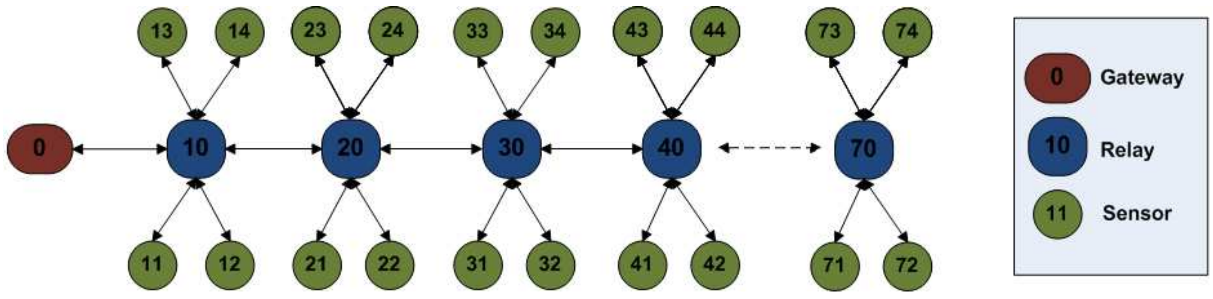


Figure 21 - Network topology and node addresses

The original protocol specified that the AODV protocol used would be one taking into account the LQI as well as the number of hops to draw a route between two nodes. This way, the node would choose a route with better quality and more hops rather than a route with less hops but a very bad link quality.

In the simulation I conceived, I used the AODV protocol already in the ns2 simulator which does not take into account LQI. Therefore, if node 31 wants to transmit its temperature packet, it will first send it to its relay, node 30. Then, node 30 will use AODV protocol to construct the route to node 0. AODV will respond that node 30 have to send the message to node 10 and then 0. But we can wonder if sending it first to node 20, then 10 and finally 0 would give better results because of a better LQI or not.

5.5. Experimentation results

In this part, I will study the influence of some parameters on the useful bitrate, loss rate or the energy consumption.

5.5.3. Useful bitrate and loss rate vs Distance

Conditions:

Here, I will test the influence of the distance on bitrate and loss rate.

All seven relay nodes will be started in order to perform AODV and conduct the messages from the sensor node to the gateway. However, only one sensor node will be started, the one we want to analyze the useful bitrate and the loss rate. For example, for a distance of 20 meters from the gateway, only node 11 will be started and for a distance of 80, only node 41 will be started.

The number of vibration packet required is 100 and each packet has a size of 100 Bytes.

The useful bitrate is computed from the reception of a temperature packet by the gateway to the end of all vibration packets reception.

Expected results:

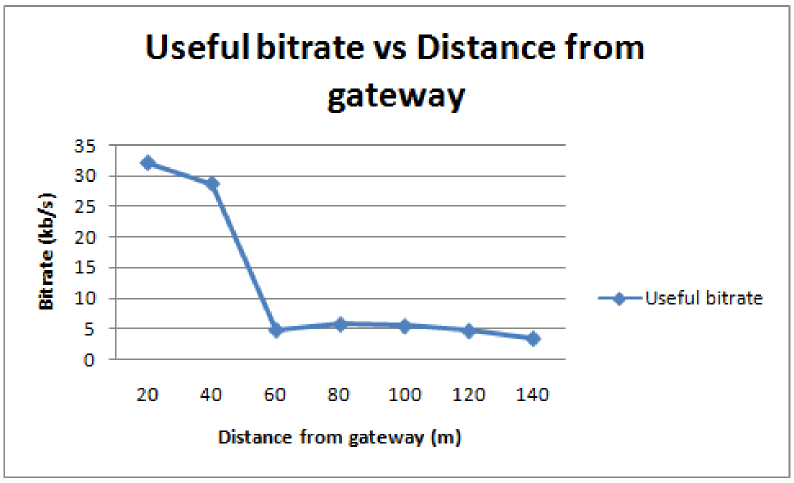
From this experiment, we expect both the useful bitrate and the loss rate to decrease with distance. Indeed, from a large distance, packets will need more time to travel across the train to the gateway. Moreover, the packet will have to pass through various relays, increasing the odd to be lost.

Obtained results:

Table 39 shows the numerical results for the useful bitrate and the loss rate versus the distance to the gateway. Graphic 15 displays the results of the useful bitrate on a graph whereas Graphic 16 displays the results of the loss rate.

Useful birate and loss rate vs Distance		
Simulation		
Distance (m)	Bitrate (kb/s)	Loss rate (%)
20	32,00	47,30
40	28,60	45,30
60	4,88	64,00
80	5,80	60,85
100	5,50	66,50
120	4,73	66,99
140	3,49	72,10

Table 39 - Useful bitrate and Loss rate vs Distance

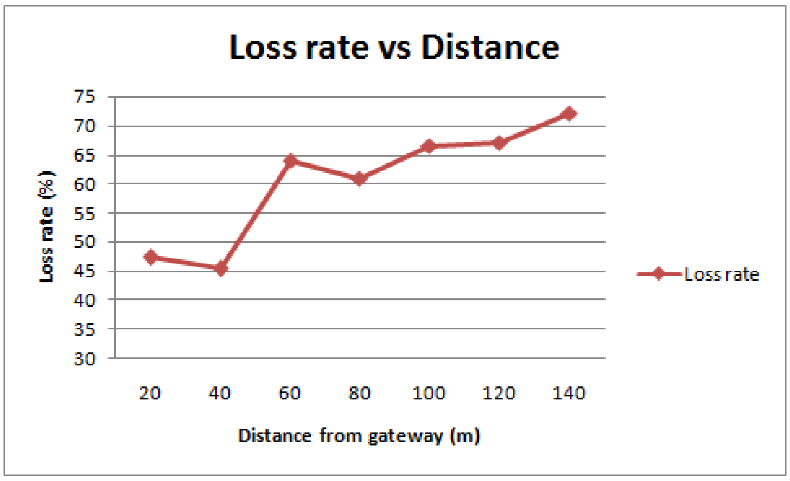


Graphic 15 - Useful bitrate vs Distance

We can see that the useful bitrate decreases when the distance from the gateway increases. That's what was expected. However, between a distance of 40m and a distance of 60m, the bitrate loses about 24 kb/s. That loss is very high and is not explained. From 80m to 140m, the slope decreases slowly and does not show the same plateau as before whereas the number of hops to get to the gateway increases.

By analyzing the packets drop, I saw that the majority of the packets are dropped on node 20 situated at 40m. This node has exactly the same configuration as the others and there is no reason it should drop so much packets.

To conclude, while we can see that the tendency of the slope is the one expected, there seems to be a problem with large distance. The useful bitrate for a distance higher than 40 meters is then very low, only about 5 kb/s.



Graphic 16 - Loss rate vs Distance

We can say that loss rate tends to what was expected for its variation with distance. However, the value of the loss rate is very high. A loss rate higher than 50% does not seem normal. Maybe the simulator drops more packets than it should.

5.5.4. Useful bitrate and loss rate vs Packet size

Conditions:

This experiment will serve to analyze the influence of vibration packet size on useful bitrate and loss rate.

Here, only node 11 will be started to keep interferences low and distance constant.

The number of vibration packets sent remains 100 but their size will vary from 10 bytes to 100 bytes.

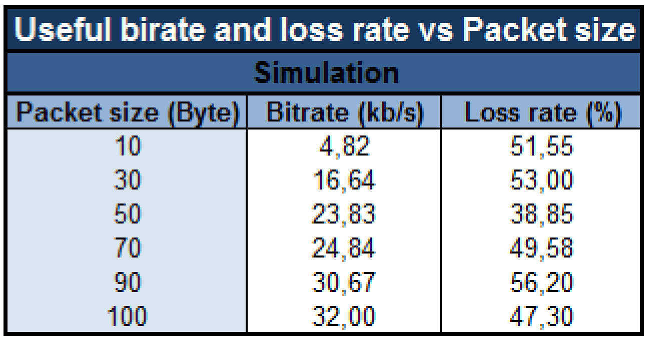
Expected results:

From this experiment, it is expected that the useful bitrate increases with packet size. Indeed, the impact of headers or CSMA is larger when the payload length is small.

We can also expect that the loss rate would be higher with larger vibration packets.

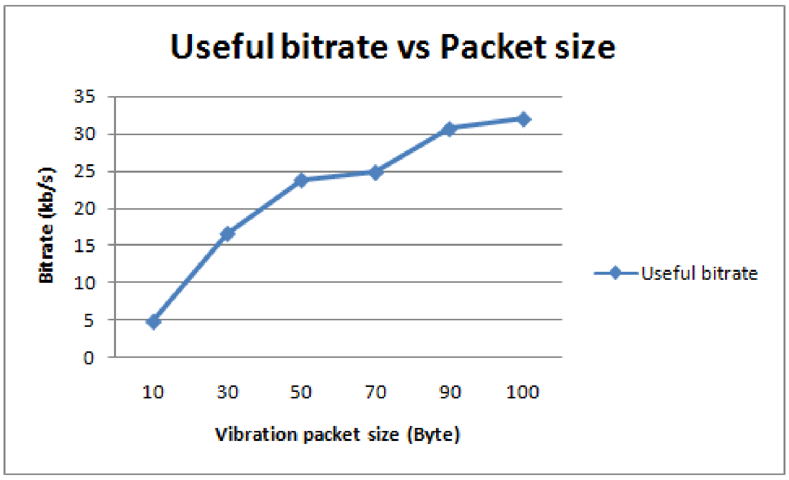
Obtained results:

Table 40 shows the numerical results of useful bitrate and loss rate varying with vibration packet size. Graphic 17 shows the graph of the variation of useful bitrate and Graphic 18 displays the graph for the loss rate.



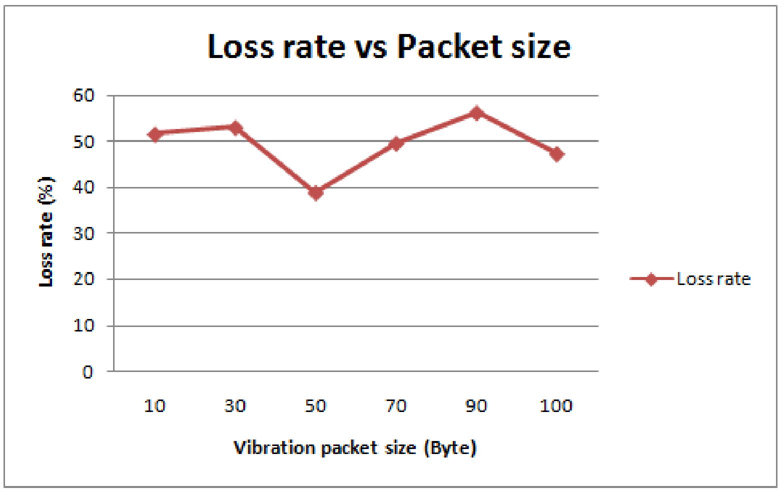
Useful birate and loss rate vs Packet size		
Simulation		
Packet size (Byte)	Bitrate (kb/s)	Loss rate (%)
10	4,82	51,55
30	16,64	53,00
50	23,83	38,85
70	24,84	49,58
90	30,67	56,20
100	32,00	47,30

Table 40 - Useful bitrate and loss rate vs Packet size



Graphic 17 - Useful bitrate vs Packet size

The variation of the useful bitrate versus the vibration packet size is exactly what was expected, it increases with the size.



Graphic 18 - Loss rate vs Packet size

The curve of the loss rate does not seem to show any tendency to increase or increase. This is not at all what was expected. It almost appears that there is a more or less fixed value of about 50% losses and that the length of the packets has no influence at all.

5.5.5. Energy consumption vs Network size

Conditions:

The purpose of this experiment is to emphasize the impact of the network size, that is, the number of nodes in the network, on the sensor's lifetime.

For this test, the vibration packets size is fixed to 100 bytes and the number of packet is 100. The number of sensors started varies from 1 to 28. We will however focus on the energy consumption of node 11.

If the sensor works with two AA batteries, it will work on 3V and have 2000mAh. Converting it on Joules, it gives 21,600 Joules.

From the remaining energy of the node after a fixed amount of time and given that it had an initial energy of 100 J, the expected lifetime of the sensor can be computed.

Expected results:

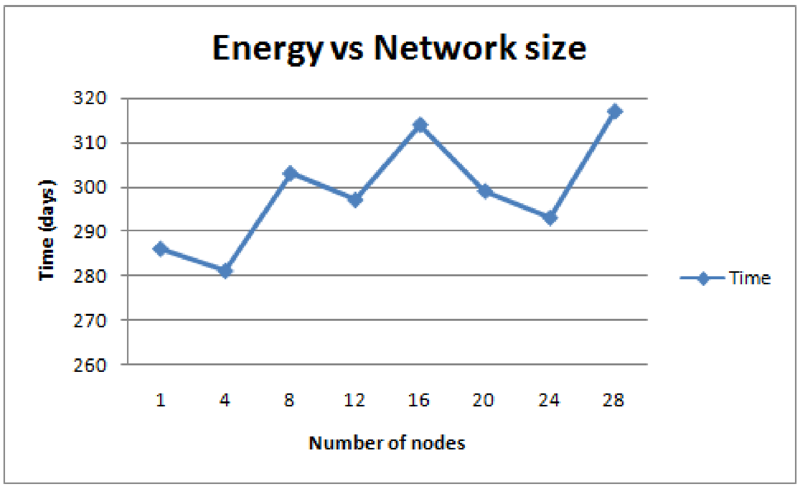
It is expected that the sensor lifetime would increase with the size of the network. Indeed, we can assume that the major part of the energy consumption takes place during the transmission of the 100 vibration packets. When the network is composed by a lot of sensors, node 11 is not able to transmit its vibration packets as often as it would do if it were alone. Indeed, the sensors transmit their vibration information in a round robin fashion.

Obtained results:

Table 41 shows the numerical results for the longevity of the sensor depending on the network size. Graphic 19 shows the graph.

Energy vs Network size	
Simulation	
Network size	Time (day)
1	286
4	281
8	303
12	297
16	314
20	299
24	293
28	317

Table 41 - Lifetime vs Network size



Graphic 19 - Lifetime vs Network size

The longevity of the sensor tends to increase with the number of nodes in the network. However, it is not a regular increase; it has some variation, going down before it rises again. This can be due to the fact that for this experiment, I couldn't repeat it several times. Indeed, for each case, I could only run the program once because the randomness of the simulator is random during one run but if we want to rerun the program, the same sequence will be played again.

This experiment tells us that the sensor can last at least 10 months before it runs out of power.

Conclusions

The results obtained by the simulation of the train network can give some perspectives on the results we could expect from a real deployed network.

However, I find there are a lot of inconsistencies between the expected results and the obtained ones.

Taking into account the characteristics of the simulator we wanted, NS2 was the only choice. Though, given my experience on NS2 I acquired during this project, I can point out various drawbacks. First of all, the simulator itself is not very user friendly. To simulate a simple network can take some time at the beginning. But when we want to do something more complex as I had to do, develop our own application layer for example, this is where it really becomes troublesome.

I found the ns2 simulator not very stable. Quite often, when I was working on the application level, it raised problems on a totally different level which was not at all connected to the changes made. In this case, it is very difficult to find where the problem comes from and hours can be spent working with the debugger.

Another remark is that I just can't say if the inconsistencies seen in the simulation come from flaws from the simulator itself or from the code I made. I know that my program is not perfect and that there are flaws which can impact the results. But maybe flaws on the lower layers of the simulator exist also and could explain for example the very high rate of packet loss.

I think that the new NS3 simulator coming up could be a good alternative when it is finished. Indeed, more reflection has been conducted on the conception of the simulator and it would less be patch codes as it is a little bit with ns2.

References

1. **GloMoSim.** About GloMoSim. [Online] http://pcl.cs.ucla.edu/projects/glomosim/.

2. **UCLA Parallel Computing Laboratory.** UCLA Parsec Programming Language. [Online] http://pcl.cs.ucla.edu/projects/parsec/.

3. **GloMoSim.** Obtaining GloMoSim. [En ligne] http://pcl.cs.ucla.edu/projects/glomosim/obtaining_glomosim.html.

4. **OPNET.** *Application and Network Performance with OPNET | Monitoring, Troubleshooting, Auditing and Prediction.* [Online] http://www.opnet.com/.

5. —. University Program-Research requirements. *OPNET.* [Online] http://www.opnet.com/university_program/research_with_opnet/research_requirements.html.

6. —. Untitled Document. [En ligne] http://www.opnet.com/4d_forms/university/app_research_w.html.

7. —. OPNET solutions. [Online] http://www.opnet.com/solutions/index.html.

8. **OMNeT++ Community.** *OMNeT++ Community Site.* [En ligne] http://www.omnetpp.org/.

9. —. Downloads | OMNeT++. [En ligne] http://www.omnetpp.org/component/docman/cat_view/1-omnet-releases.

10. Nsnam. [En ligne] http://nsnam.isi.edu/nsnam/index.php/Main_Page.

11. OTcl. [En ligne] http://otcl-tclcl.sourceforge.net/otcl/.

12. ns2 Class Hierarchy. [En ligne] http://www.isi.edu/nsnam/nsdoc-classes/hierarchy.html.

13. Nam: Network Animator. [En ligne] http://www.isi.edu/nsnam/nam/.

14. **SourceForge.** Browse nsnam Files on SourceForge.net. [En ligne] http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.34/.

15. Low Rate Wireless Personal Area Networks - NS2 Simulation Platform. [En ligne] http://www-ee.ccny.cuny.edu/zheng/pub/index.html.

16. The ns-3 network simulator. [En ligne] http://www.nsnam.org/.

17. **doxygen.** NS-3. [En ligne] http://www.nsnam.org/doxygen-release/index.html.

18. **Source Forge.** Brows nsnam Files on SourceForge.net. *sourceForge.net.* [En ligne] http://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.34/.

19. **Free Software Foundation.** GDB: The GNU Project Debugger. [Online] http://www.gnu.org/software/gdb/.

20. —. DDD - Data Display Debugger - GNU Project. [Online] http://www.gnu.org/software/ddd/.

21. **Eclipse.** Eclipse Downloads. [Online] http://www.eclipse.org/downloads/.

22. *802.15.4 standard.*

23. **IEEE Computer Society.** *IEEE std 802.15.4 .* 2006. p. 70.

24. —. *IEEE std 802.15.4.* 2006. p. 95.

25. —. *IEEE std 802.15.4.* 2006. p. 107.

26. —. *IEEE std 802.15.4.* 2006. p. 107.

27. —. *IEEE std 802.15.4.* 2006. p. 113.

28. —. *IEEE std 802.15.4.* 2006. p. 125.

29. —. *IEEE std 802.15.4.* 2006. p. 130.

30. **Latré, Benoît, et al.** Maximum throughput and Minimum delay in IEEE 802.15.4. [éd.] Springer Berlin / Heidelberg. *Mobile Ad-hoc and Sensor networks.* 2005, Vol. 3794/2005, pp. 866-876.

31. **Lee, Jin-Shian.** Performance Evaluation of IEEE 802.15.4 for Low-Rate Wireless Personal Area Networks. *IEEE Transactions on Consumer Electronics.* August 2006, Vol. 52, 3, pp. 742-749.