Nom : Moctar Ba
Filière : SICOM
Année scolaire : 2015 – 2016
Institution : INRIA (Institut national de recherche en informatique et en automatique)

Monitoring sur plateforme IoT-Lab

Maître de stage : Frédéric Saint-Marcel
Courriel : frederic.saint-marcel@inria.fr

Contents

Introduction

INRIA (Institut national de recherche en informatique et en automatique) is a French national research institution focusing on computer science and its applications. I spent my internship in the SED (Service d'Experimentation et de Développement), which goal is to assist researcher by setting up experiments and developing software solutions.
I joined the team responsible for the IoT-Lab: an experimental plateform for researchers to run experiments on the **Internet of Things**.

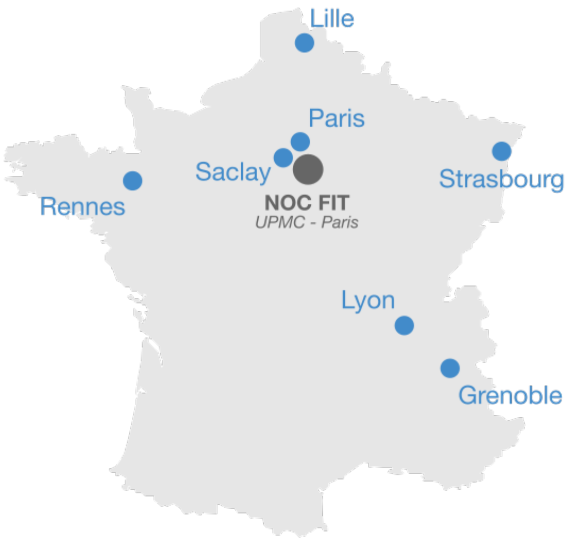
« *The internet of things (IoT) is the internetworking of physical devices, vehicles, buildings and other items—embedded with electronics, software, sensors, actuators, and network connectivity that enable these objects to collect and exchange data.* » Wikipedia

I worked on the monitoring tools of the platform. Each embedded device in the IoT network can be monitored by an external device: however, the clock of the monitoring device is not synchronized correctly and large time differences can appear. This is a serious problem as knowing exactly when an event occured in a network is essential.

This report contains an overview of the Iot-lab platform, a more in depth explanation of the problem, and lists the solutions used to solve it.

I . IoT-lab : a large scale testbed

IoT-lab (www.iot-lab.info**)** is a large scale network of wireless **sensors nodes** equiped with tools to experiment on them. There **2728 wireless sensor nodes** located at six differents sites in France : Inria Grenoble (928), Inria Lille (640), ICube Strasbourg (400), Inria Saclay (307), Inria Rennes (256) and Institut Mines-Télécom Paris (160). Most nodes are in a fixed location, however nodes can be located on a moving robot.



Each site has it’s own nodes with an unique topology : the purpose is to enable users, mainly researcher and companies, to experiment on a physical sensor network. Complete access to each node is given : users can develop and flash new firmwares on the nodes, monitor nodes energy consumption, temperature, radio communication and network-related metrics. The facility offers quick experiments deployment, along with easy evaluation, results collection and analysis. Defining complementary testbeds with different node types, topologies and environments allows for coverage of a wide range of real-life use-cases.

A robot transporting nodes in Lille

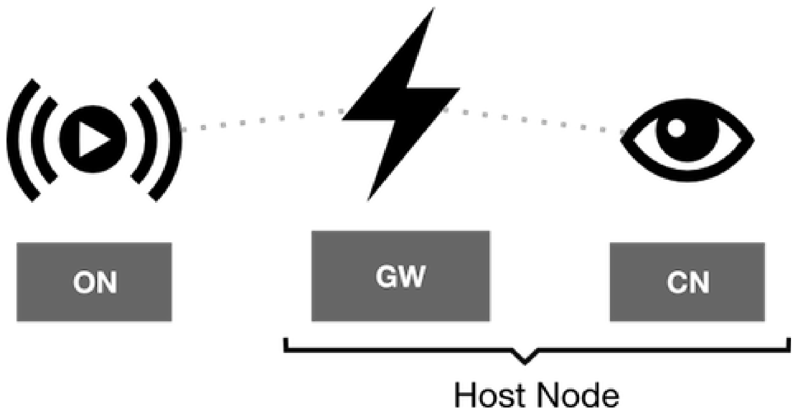
IoT-LAB is part of the FIT experimental platform, a set of complementary components that enable experimentation on innovative services for academic and industrial users. The project gives French Internet stakeholders a way to experiment with mobile wireless communications, both on network and application layers, thereby accelerating the design of advanced networking technologies for the Future Internet

II . What is an IoT-Lab node ?

1. General view

A **node** is simply an embedded computer with sensors that allows it to retrieve measures from the environnement and a radio chip for communications. Just like any other computer, it is mainly comprised of one or two microprocessors. The microprocessors used are the ARM Cortex M3 and ARM Cortex A8. Each node comes with a radio interface and standard sensors.

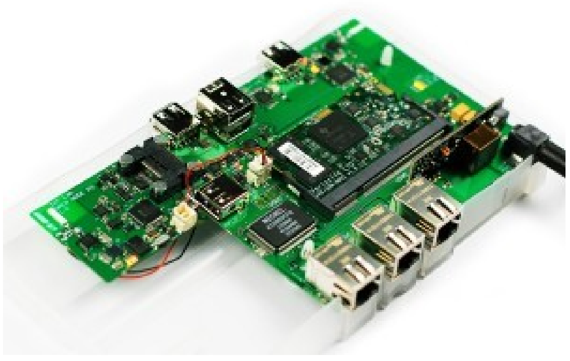
An IoT-LAB node consists of three main components : an **open node**, a **gateway** and a **control node**. The gateway and control node are part of the host node : they are not accessible by the user.



The **open node** is fully open and the user is granted a full access to the memory. This implies that he can load and run any operating system. This feature is handled using a remote access to reboot and (re)load any firmware on any node.

The **gateway** offers a connection to the global IoT-Lab servers and infrastructure (network).

The **control node** is used to interact, passively or actively, with the Open Node. It monitors the consumption and the radio activity (RSSI) of the open node during experiments and selects power supply (battery or PoE).


Node board (gateway + control node)

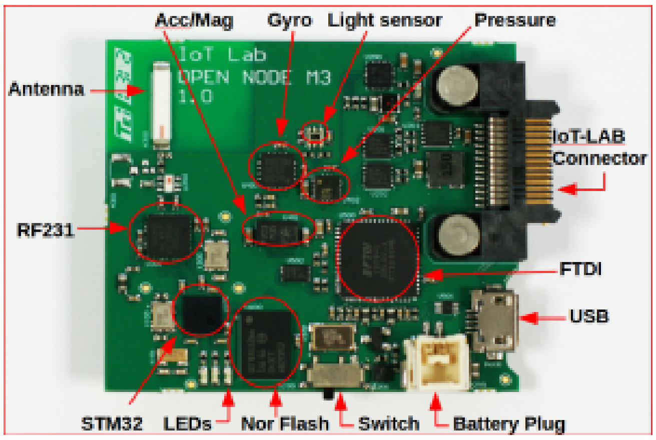
There are two generations of nodes : the **M3 node**, named after it’s main microprocessor (cortex M3) and the **A8 node** (cortex A8).

2. M3 node

The **M3 node** contains one microprocessor : an ARM Cortex-M3. The Cortex-M3 is not powerful enough to run high-level operating systems but it can run embedded operating systems like FreeRTOS and Contiki.
Cf. Annex « *Architecture of an M3 node* » for more details.

Data Sheet of the M3 node

MCU	ARM Cortex M3, 32-bits, 72 Mhz, 64kB RAM – ST2M32F103REY
sensors	• Ambient sensor light – ISL29020 • Atmospheric pressure and temperature – LPS331AP • Tri-axis accelerometer/magnetometer – L3G4200D • Tri-axis gyrometer – LSM303DLHC
radio communication	802.15.4 PHY standard, 2.4 Ghz – AT86RF231
external memory	128 Mbits external Nor flash – N25Q128A13E1240F
LEDs	green, red, blue
power	3,7V LiPo battery, 650 mAh – GMB 063040
Operating-Systems	FreeRTOS, Contiki, Riot



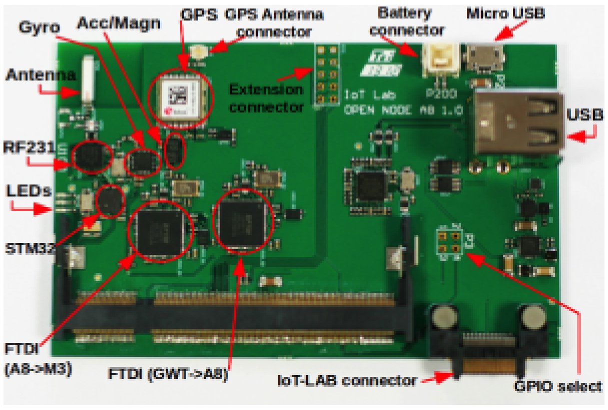
M3 Open Node design

3. A8 node

The **A8 node** is the most powerful one witch a clock speed of 600 Mhz and a memory of 256 MB. It can run high-level operating systems like embedded Linux unlike the M3 node. It contains an ARM Cortex-A8 and a co-microcontroller ARM Cortex-M3 : the Cortex-M3 gives access to sensors (tri-axis gyro and accelerometer/magnetometer), wireless communication while the Cortex-A8 runs the higher level features – flashing firmwares and accessing the M3 node serial port. Cf. Annex « *Architecture of an A8 node* » for more details.

Data sheet of the A8 node

System on Module		High-performance ARM Cortex-A8 microprocessor, 600 Mhz, 256 MB – Variscite VAR-SOM-AM35 CPU
co-microcontroller	**MCU**	ARM Cortex M3, 32-bits, 72 Mhz, 64kB RAM – ST2M32F103REY
	sensors	• Tri-axis accelerometer/magnetometer – L3G4200D • Tri-axis gyrometer – LSM303DLHC
	radio communication	802.15.4 PHY standard, 2.4 Ghz AT86RF231
	LEDs	green, red, blue
	control	USB device to control UART and JTAG – FTDI2232H
power		3,7V LiPo battery, 650 mAh – GMB 063040
option		*GPS device – MAXQ*
links		Ethernet, USB
operating-system		Linux



A8 open node design

III. Internship work Part I : monitoring tools

1. Goals

When the **control node** monitors values like radio or energy consumption, it always involves **time** : it must report a timestamp of the measure. This time must be as precise and accurate as possible because a correct ordering of events and a precise timestamp allows the user to make much better experiments and tests.

However, these timestamps are not accurate : the internal clock of the control node, which is also an M3 co-microcontroller, is not synchronized with the **gateway** time which is itself synchronized with the UTC time via NTP. The control node clock drifts at a rate of 0.4 seconds per hour which is very high considering the nodes can be up for weeks : it becomes simply impossible to have any reliable time values. Moreover, each control node seems to have it's own clock drift : ideally, all the control nodes should be perfectly synchronized during an experiment on the testbed.

The **goal** of the internship is to :

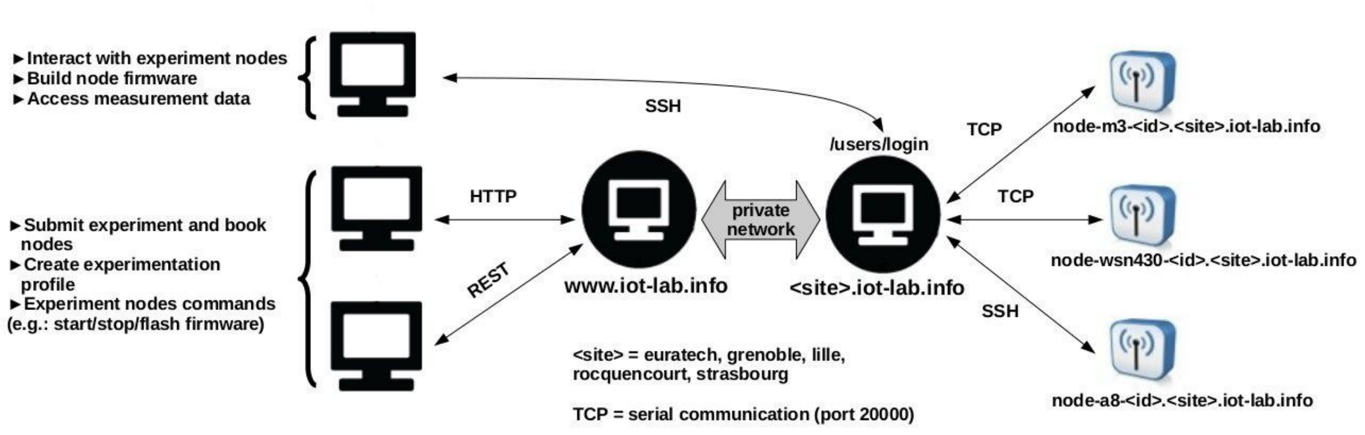
- Develop precise monitoring software that can track and record the clock drift of each control node : this allows to validate or invalidate any solution to the problem, and quantify precisely by how much the clock drift is reduced by the solution
- Assess quantitatively the clock drift at the beginning of the internship and think of a possible solution depending on the type of drift
- Implement a software solution
- Write tests for the software and validate the solution by using the previously developed tools.

All the work is done on a **test platform** (called devgrenoble site) **:** just like other sites, it contains a set of nodes. However, these nodes are not accessible to users : their goal is to allow the developers of the IoT-Lab team to test new software and modifications without altering the users experiments.

2. A first step : understanding the IoT platform as a user

At the beginning of the internship, I had to follow tutorials (https://www.iot-lab.info/tutorials/) on how to use IoT-Lab : these are the same tutorial any researcher would follow if he wanted to learn how to use IoT-Lab. This step is very important : understanding the user interface allows to make better decisions when modifying the back end code.

A user can submit an experiment on the testbed and reserve nodes for a given time. It can be done through a web portal or a command line interface (frontend SSH) using a REST API. The **gateway server**, different from the gateway of a node, is a Linux server that has access to the nodes of the site : it is a **frontend** from which the nodes can be controlled. For example, users can interact with the serial port of a node and access **monitoring data** through the frontend. The monitoring data is generated by the gateway of the node on the frontend with an OML library. OML is a file type and a measurement library (https://mytestbed.net/projects/oml) for storing raw data. It uses its own XML format.


►Interact with experiment nodes
►Build node firmware
►Access measurement data
►Submit experiment and book nodes
►Create experimentation profile
►Experiment nodes commands (e.g.: start/stop/flash firmware)
SSH
HTTP
REST
private network
/users/login
www.iot-lab.info
<site>.iot-lab.info
TCP
TCP
SSH
node-m3-<id>.<site>.iot-lab.info
node-wsn430-<id>.<site>.iot-lab.info
node-a8-<id>.<site>.iot-lab.info
<site> = euratech, grenoble, lille, rocquencourt, strasbourg
TCP = serial communication (port 20000)

Architecture of an IoT-Lab site

My first task was to to write a **python script**, running on the SSH frontend, that made the running open node firmware automatically ask to every control node it's time every second and then compared this time to the time of the frontend SSH. This tool showed that the drift was around 0.4 seconds per hour, however there were two small problems that made it slightly inacurrate. First, there is a delay between the moment the scripts asks a node for it's current time, and the moment the scripts receives the time. Second, the time of the frontend, which is used as a basis to compute the clock drift of each control node, is itself inaccurate because it uses a time synchronization protocol called NTP that does not allow to reach the levels of precisions wanted.

3. A second step : setting up the development environment and getting used to it

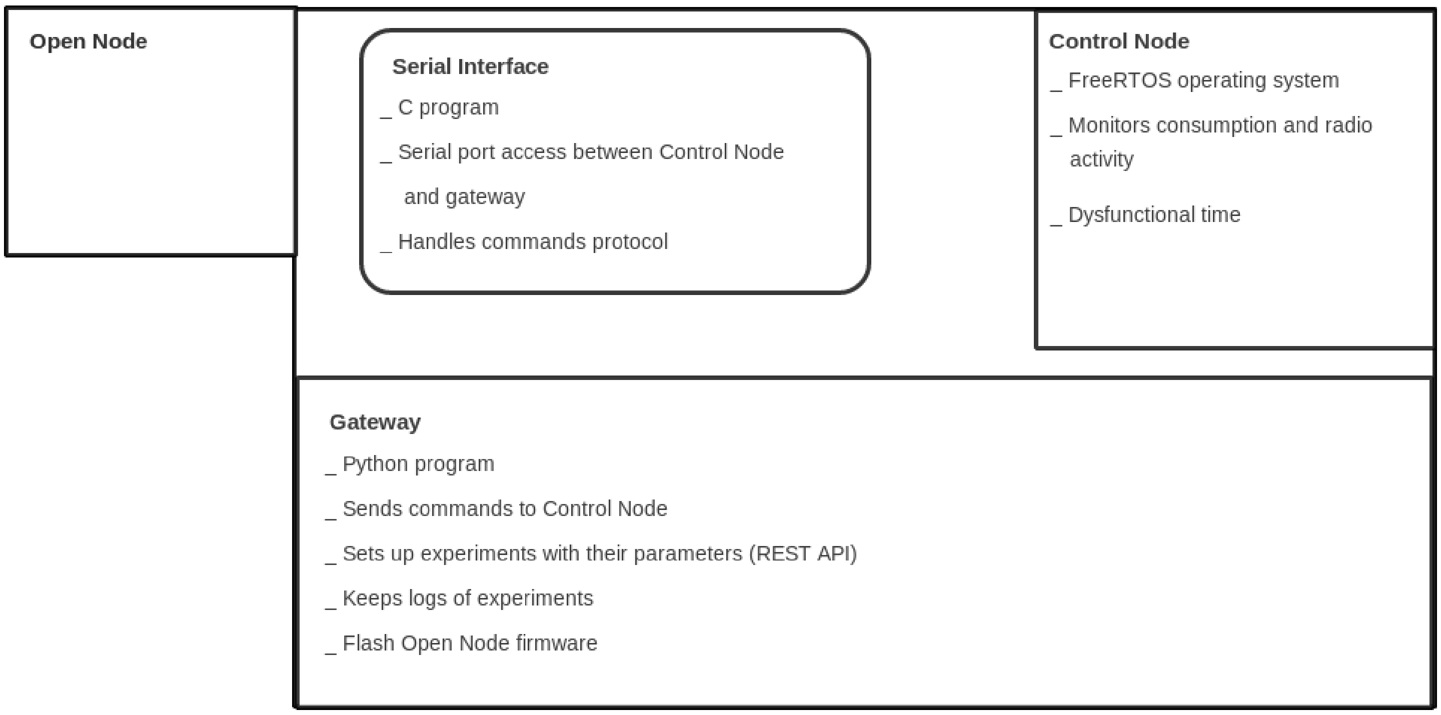
The IoT-Lab team uses a development environment centered around Linux and git.

Linux is self-exlpanatory : it is an open source, secure and highly efficient operating systems well suited for development endeavors.

Git is the version control software : the source code of all the Iot-Lab projects is stored in a central repository accessible from a private repository in **Github**. Only Iot-Lab approved member can access the source code. **Git** allows to get a local copy of the repository, make local modifications and submit them while keeping track of the history of changes. It is then easy to come back to an earlier version if a bug or mistake happens.

Testing is another important part of the environment : new code must be thoroughly tested to ensure it works correctly. I had to learn how to write test code and check that they cover all the appropriate cases.

Iot-Lab uses many **programming langages** but I mainly worked with **C** and **Python**. I already knew these langages so I didn't have to learn them from scratch. C is the langage used for the operating system of the control node : most of the work I did was to modify this operating system. It is also used for the serial interface : a program that defines a communication protocol via a serial port. Python is mostly used for the gateway code, monitoring tools and scripts between the gateway and the control node.


Open Node
Serial Interface
_ C program
_ Serial port access between Control Node and gateway
_ Handles commands protocol
Control Node
_ FreeRTOS operating system
_ Monitors consumption and radio activity
_ Dysfunctional time
Gateway
_ Python program
_ Sends commands to Control Node
_ Sets up experiments with their parameters (REST API)
_ Keeps logs of experiments
_ Flash Open Node firmware

IoT-Lab node software architecture

4. A third step : writting the monitoring software

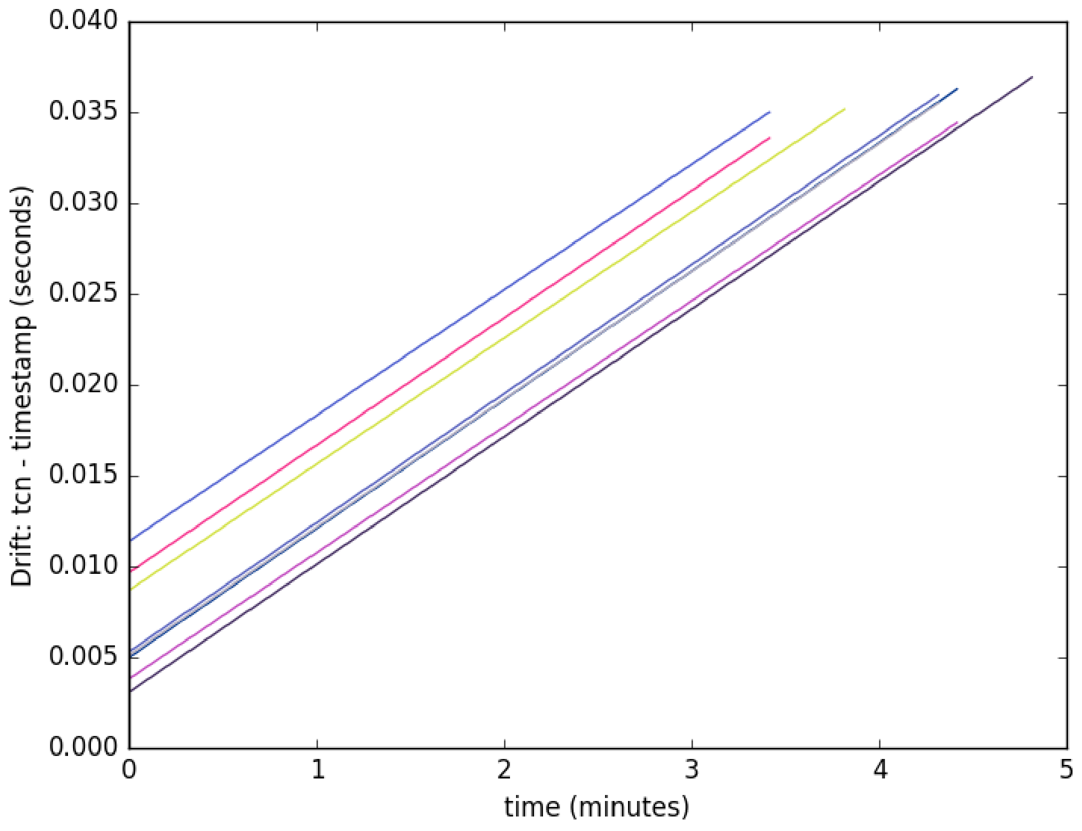
Before I got to dive on the control node operating system source code, I had to develop a monitoring software to make a precise quantitative assesment of the clock drift : the goal was to correct the flaws of the first python script.

The second version involved more work but corrected these two issues with a single solution : the use of a **GPS on** A8 nodes. A GPS can give the current time with near perfect precision. Since some nodes have a GPS connection, I used them to get a precise measure of the clock drift. I had to modify the control node **operating system** so that it would active the GPS PPS (pulse par second) which is a signal sent by the GPS every second. It sends a packet containing the node's time each time a GPS signal is received to the gateway. Then the raw data from these packets is extracted and saved in an OML file. The packets were sent to a serial port and processed by a software called **serial interface**. I modified the interface so it could support this new kind of packet and store them in a file so that the clock drift data could be processed.

The last component of the monitoring software is a python script executed on the SSH frontend that **processes** the clock drift data of each node : the script takes as input the raw file that contains the data from the received packets. Then it computes the clock drift of each node and some statistics about it before plotting the clock drift as a function of time.

Once the tool was ready, it was run on the A8 nodes of the devgrenoble site : a set of nodes only accessible to the IoT-Lab team for testing purposes. The goal was to learn more about the shape and characteristics of the clock drift before trying to find a fix.

Quantitative analysis of the clock drift on each A8 node on the devgrenoble site :



The clocks of the A8 nodes each have a time drift of around 117 ppm (microsecond per second). This drift is much higher than the expected drift of an ordinary quartz clock (around 1 ppm per °C).

Some statistics on the clock drift:
Node name: drift (ppm)

A8-60: 115.541484716
A8-53: 118.041509434
A8-56: 117.355212355
A8-61: 118.297297297
A8-50: 116.712195122
A8-59: 115.07804878
A8-54: 117.107266436
A8-62: 115.505660377

Mean: 116.704834315
Std: 3.21643711418
Spread: 2.75604445442%

IV. Internship work Part II : correcting the clock drift

1. Correcting the constant clock drift

As seen in the clock drift statistics, the drift has two components: a constant one of around 120 ppm common to all the clocks and another drift between each M3-node clock with a spread of around 6 ppm.

The M3 nodes come with a complete **time library.** Before correcting the clock drift, this time has to be updated and fixed as it has a few bugs and is not efficient. It is written with very low level code that does a few things: it picks up the processor timer, transforms it into a real time, handles the timer overflows and configures a quartz frequency among other things. The code was written for 32 bits and had to be modified to handle 64 bits timers as they overflow less frequently.

The time library works this way : a quartz frequency is configured at 32678 Hz and a **timer tick** count is established. This means that the timer tick goes up by 1 every $\frac{1}{32768}$ second. It is then possible to compute the time in seconds by using the following formula.

$$Time = \frac{Timer\,ticks}{Quartz\ Frequency}$$

However, the configured quartz frequency still relies on the physical frequency of the M3-nodes processors which is 72Mhz. Since the configured quartz has a frequency of 32678 Hz, the processor frequency is not a multiple of the quartz frequency. Hence, every time we make an integer divide with the two (to update the timer tick), a small error is made that creates the constant drift:

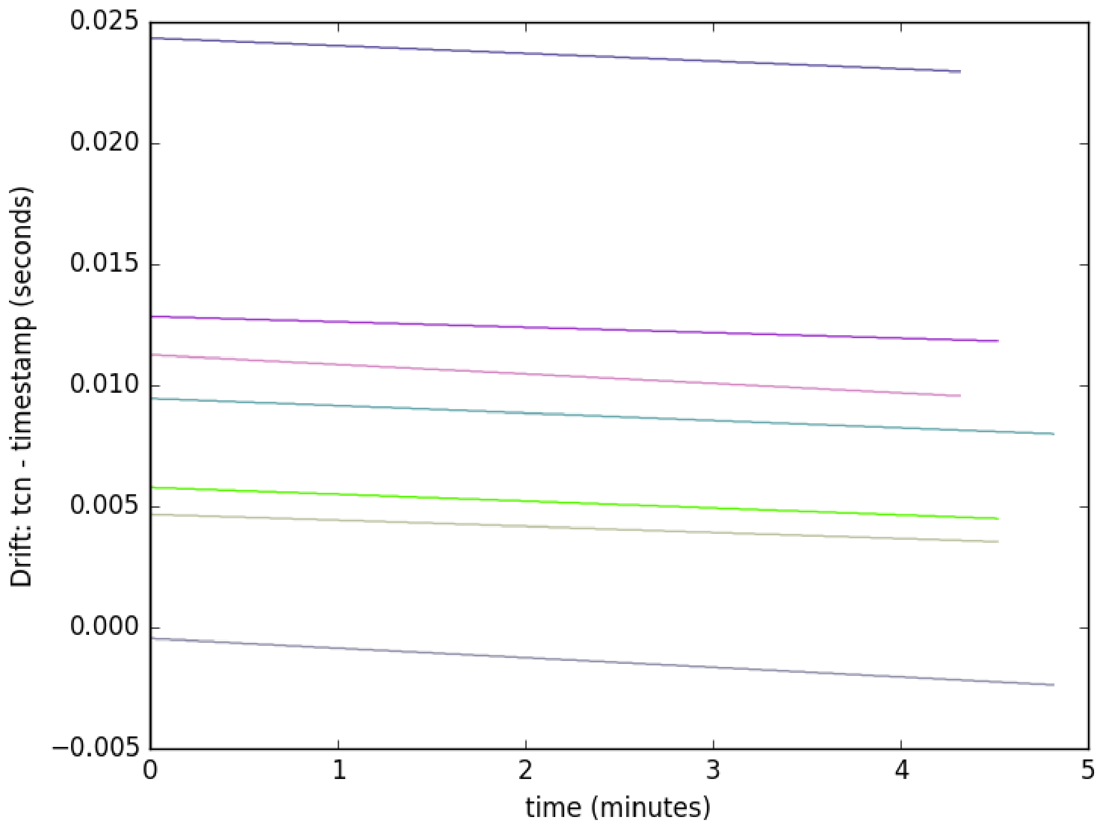
$\frac{72\,000\,000}{32678} = 2197$ (Integer divide)

$\frac{72\,000\,000}{32678.} = 2197.265625$ `(float divide)`

So in reality, the clock behaves as if it's frequency is:
32768 * (72000000 / 32768.) / (72000000 / 32768) = 32771.96176604461

This can be fixed with a workaround: changing the Quartz Frequency in the Time formula by dividing the timer tick with 32772 (a close approximation of 32771.96176604461) instead of 31678. Once this is done, we get an error of 5 ppm instead of 120 ppm. We can further reduce the ppm by using a more precise approximation of 32771.96176604461. In the time library, no float or double elements are used because the embedded processor does not have a floating point unit unlike desktop processors. This means that floating point operations are expensive : it is better to use an integer.

Drift of A8 nodes after correction of constant drift



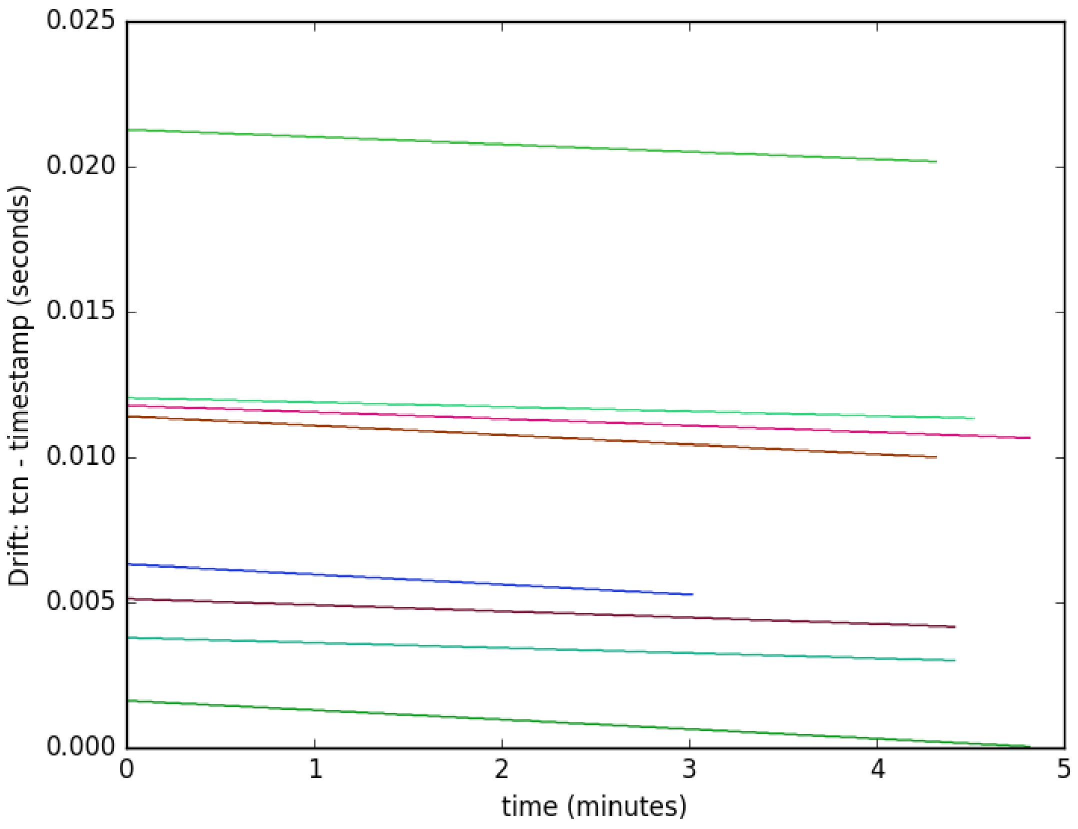
Drift microseconds / second:

A8-62: -6.65397923875
A8-54: -5.06920415225
A8-60: -6.4787644787
A8-56-: -4.84132841328
A8-53: -4.16605166052
A8-61: -3.71586715867
A8-50: -5.30115830116

Mean: -5.17519334334
Std: 2.67818445026
Spread: -51.7504230775%

The error is reduced by around 120 ppm as predicted. Next, a more precise frequency is implemented : the time library is modified so that it can use a frequency of 32771.962 instead of 32772. To do this without using a float, we introduce an intermediate variable which value is 32 771 962 and multiply by 1000 the timer before each division. The end results is the same as if the frequency was 32771.962 but no floating point operation is made.

Clock drift on A8 nodes with a more precise theoretical frequency



Here we have the clock drift on A8 nodes after we use the more accurate theoretical frequency of 32771.962 Hz The mean drift is now -4.25 ppm which is better than -5.17 ppm obtained by using the rounded frequency of 32772 Hz. However, the mean drift is still high: the next step is to compute a practical frequency which puts this mean drift closer to zero.

Drift microseconds / second:

A8-54:	-3.90657439446
A8-61:	-2.66789667897
A8-53:	-2.98867924528
A8-60:	-5.32046332046

A8-62: -5.49134948097
A8-56: -3.67924528302
A8-59-: -5.75138121547
A8-50: -4.25868725869
Mean: -4.25803460967
Std: 3.07867307384
Spread: -72.3026784904%

2. Use of a better frequency

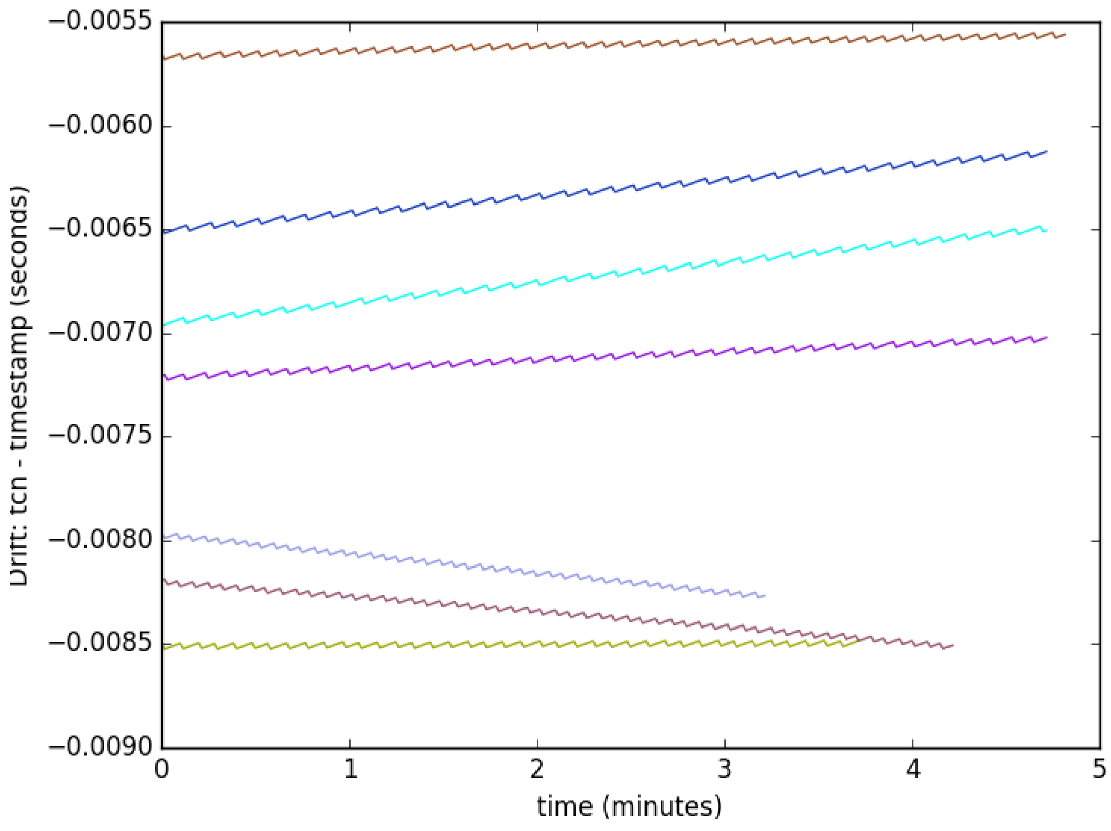
After assessing the mean drift of around 5 ppm with the theoretical frequency of 32771.962 Hz, we adjust it to further reduce the mean drift. Since we want a mean drift of 0 ppm, we can use the following formula to compute what frequency we should use to obtain it :

$$Correct\,Frequency = Incorrect\,Frequency \times (1 - drift\ per\ second)$$

With an Incorrect Frequency of 32771.962 and a correspond drift per second of -4.25 ppm :

$$32771.962 \times (1 - 4.25 * 10^{-6}) = 32771.821$$

After a simple computation, we find it to be 32771.823 Hz. The mean drift is now 0.17 ppm which is much better than the original 120 ppm. The current clock drift is now very small, however we still haven’t fixed the variable drift between each node that has a standard deviation of 2.94. This is much better but still leaves us with a drift between the nodes that must be corrected with a synchronization protocol.



Clock drift microseconds / second:

A8-50: 0.0627802690583
A8-56: 0.657243816254
A8-61: 1.62544169611
A8-53: 1.3038869258
A8-59: -1.55958549223
A8-62: -1.23320158103
A8-54: 0.332179930796

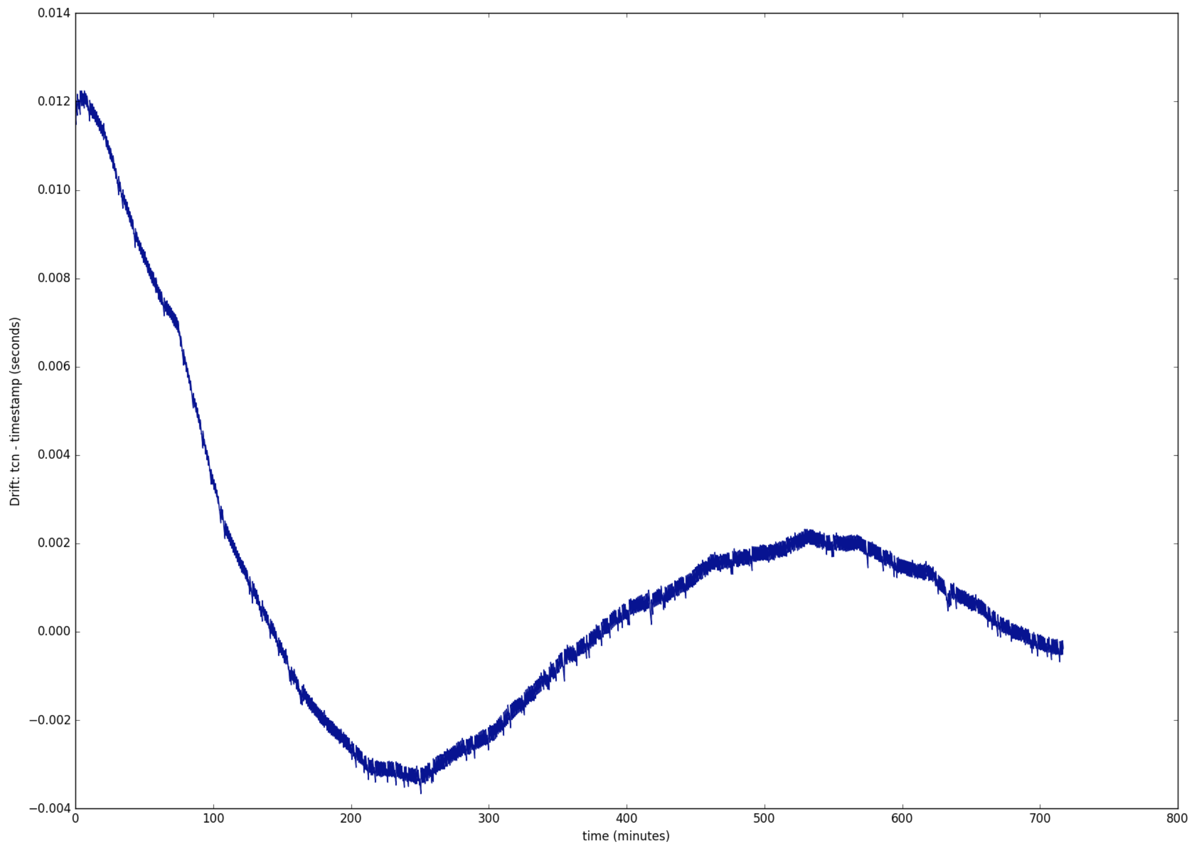
Mean: 0.169820794966
Std: 2.93932908255
Spread: 1730.84166939%

3. Use of set_time command

The following synchronization protocol is implemented : a periodic set_time command is set up from the gateway python code: every minute a set_time command with the current gateway time is sent to the control node so it can

synchronize its time with the gateway. The goal is to synchronize each control node so that the variable drift (2.94 standard deviation) is reduced.
The time library is modified once more so that the clock frequency of each control node is dynamically updated after each set_time command to correct its drift on the go : the update takes into account the error that occured since the last update and computes a new clock frequency to adjust the control node’s clock. A packet is sent from the control node to the gateway after each set_time : the packet contains the current frequency, when the last set_time operation took place and the value of the clock correction made by the set_time. The packet values are stored via OML : this allows to have a complete log of the clock behaviour during an experiment.

We can see the effect of set_time: the time drift has a periodic shape that converges toward the correct time. This is because of NTP : as said before, the gateway time is synchronized to UTC via NTP so it is expected that the control node time, which is itself synchronized to the gateway, has the shape of the NTP time.



Takeways

The goal of the internship was to develop monitoring tools for the clock drift on the control node and then fix the clock drift. In order to accomplish this task, I had to understand how IoT-Lab works and get familiar with the already existing development environment and source code. Writting new software for an already large project requires respect of coding conventions and extensive testing. I was able to write a program that automatically provided a quantitative assesment of the clock drift on each control node. Then I took steps to fix it : first by modifying the clock frequency and how it is handled and then by using a set_time command that sends the current time to the control nodes. The IoT-Lab team now has a way to monitor the time in the control node and a much better control node clock. At the beginning, the clock had a drift of 400 milliseconds per hour (120 ppm) meaning it would report an error of 9.6 seconds after only one day. Now, the mean drift is of 0.6 millisecond per hour. And with the use of the periodic set_time command, the error can be reset periodically.

The work I had to do made me learn a lot and allowed me to extensively practice my skills in software development, working and interfacing with hardware materials and efficiently using the software engineering tools that are found in most work environments.

I want to thank Frédéric Saint-Marcel, my internship tutor, for providing me with such a valuable experience. I also want to thank Gaëtan Harter for all the technical help and guidance he provided.

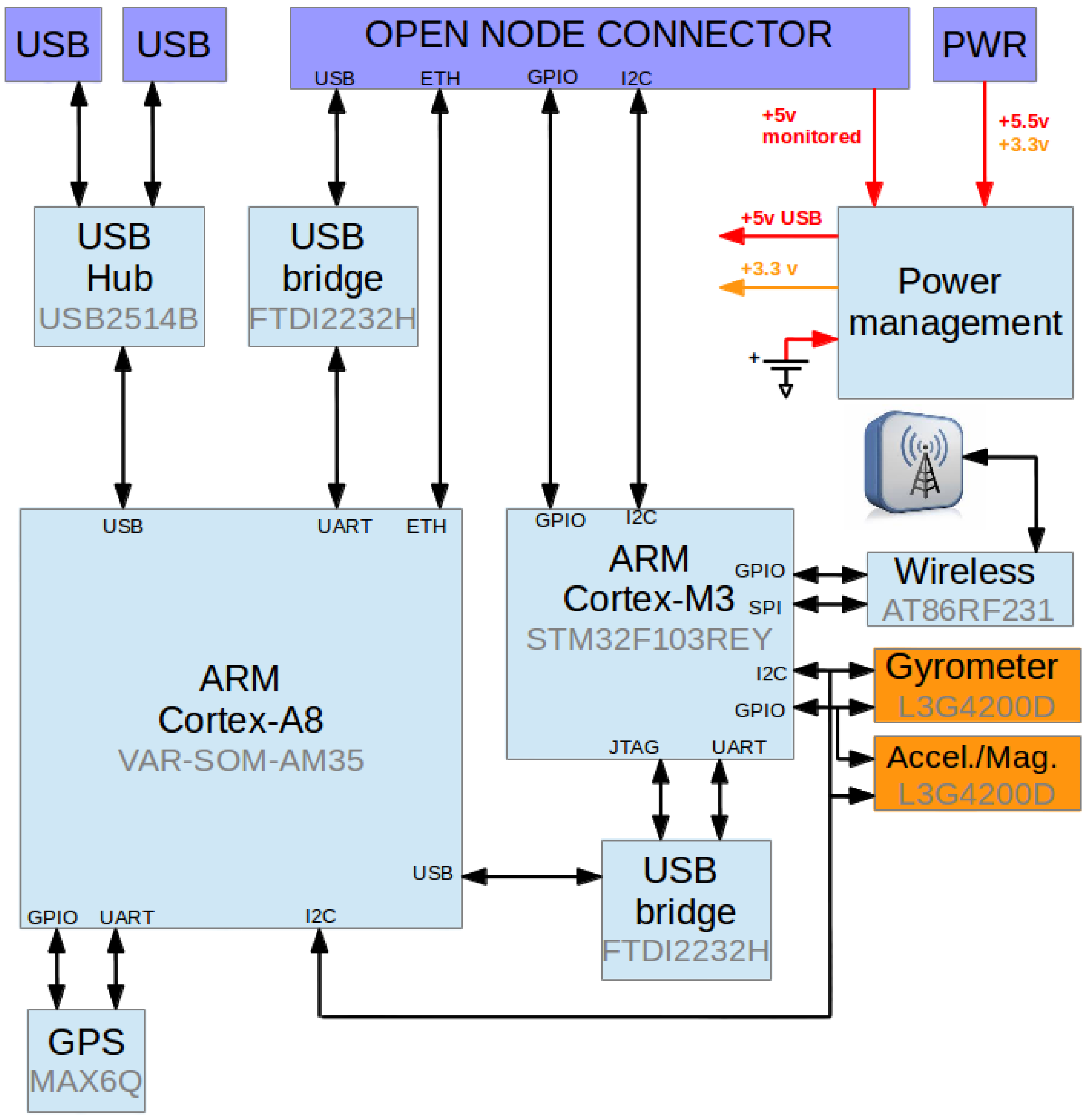
Going Forward

In this section I briefly explain some steps that could be taken to improve the work.

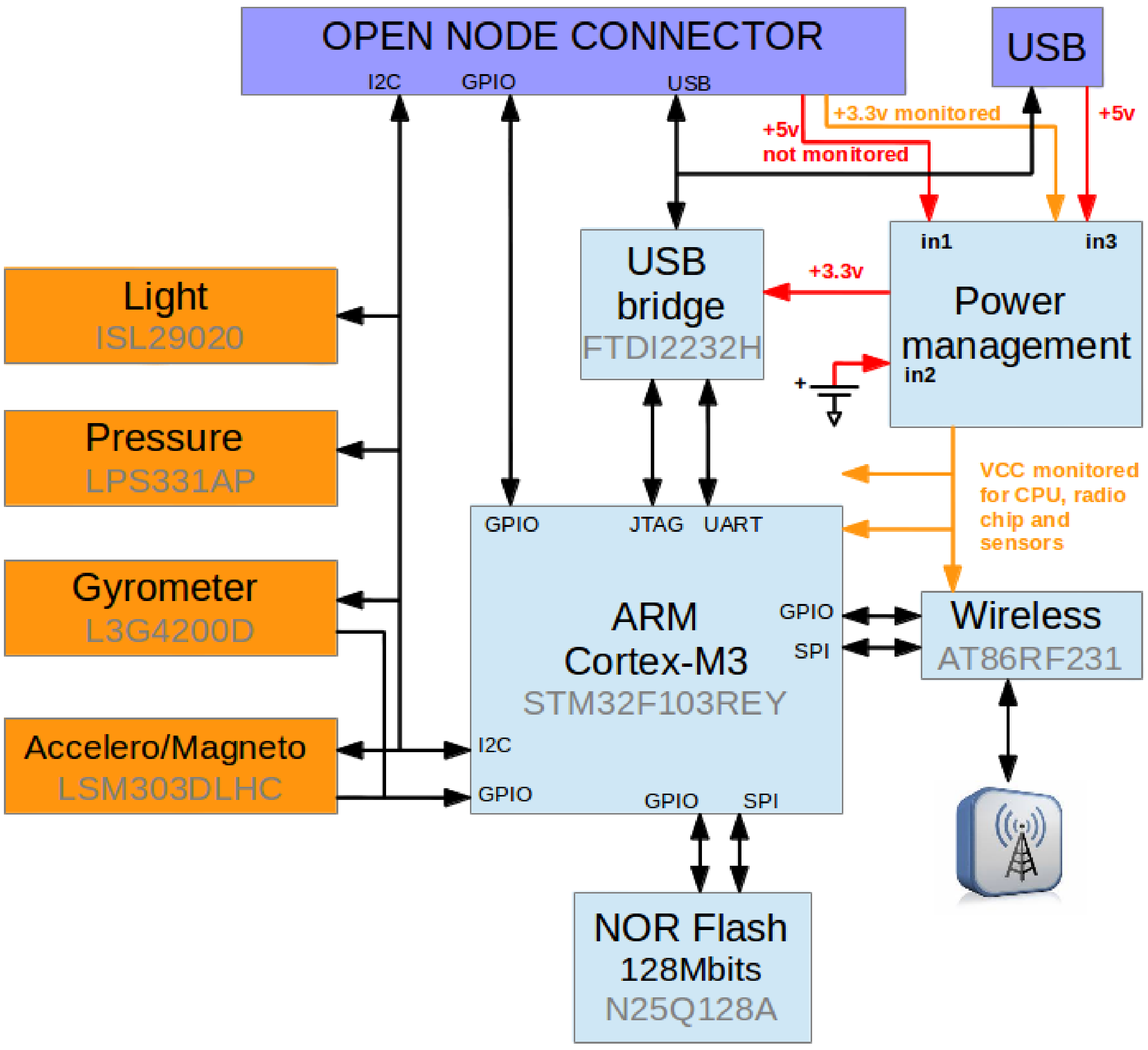
The set_time command synchronizes the time of the control node with the time of the gateway. However, the gateway uses NTP which can report a big time error when the network becomes busy. To make better use of the set_time command, it would be beneficial to use an other time protocol like PTP on the gateway.

Another possible improvement is to give NTP packets a higher priority so it is not affected by a surge of network activity.

Annex : Achitecture of an A8 node



Annex : Architecture of an M3 node



Abstract

IoT-Lab is a platform of wireless sensor nodes desgined for experimenting and testing purpose. Researchers and users can test their Internet Of Things models on a physical platform.

Each node is monitored by a device called control node : this device time is incorrectly synchronized. After developing time monitoring software, it is found that each control node has its own time drift. However, they all share a constant drift. The mean clock drift is of 120 ppm (parts per million e.g. microseconds / second) : in an hour the clock drifts by 400 milliseconds.

To correct the constant and variable drift, the time library of the control node is updated in order to correct bugs and upgrade some parts of the library to make them more reliable. The constant drift is corrected by modifying the timer frequency of the clocks while the variable drift is corrected by using a synchronization protocol that updates the time of each control node every minute. In the end, the drift is of 0.17 ppm or of 0.6 ms per hour.

Résumé

IoT-Lab est une plateforme de nœuds capteurs sans-fil mise en place à des fins d’expérimentation et de test. Chercheurs et autres utilisateurs peuvent implémenter leur modèle Internet Of Things sur une plateform physique réelle.

Chaque nœud est monitoré par un appareil appelé control node : le temps de cet appareil n’est pas correctement synchronizé. Après développement d’un logiciel pour monitorer le temps, il est remarqué que chaque control node à sa propre dérive temporelle. Cette dérive se sépare en une dérive constante qui leur est commune et d’une dérive variable propre. La dérive temporelle moyenne est de 120 ppm (partie par million e.g. microsecondes / seconde) : en une heure l’horloge dérive de 400 millisecondes.

Pour corriger la dérive, la libraire interne de temps du control node est mise à jour afin de corriger des bugs, mettre à niveau certaines parties pour les rendre plus robustes et fiables. La dérive constante est corrigé en modifiant la fréquence du timer des horloges alors que la dérive variable est corrigé en mettant en place un protocol de synchronisation qui met à jour le temps de chaque control node chaque minute. On obtient alors une dérive moyenne de 0.17 ppm our 0.6 millisecond par heure.