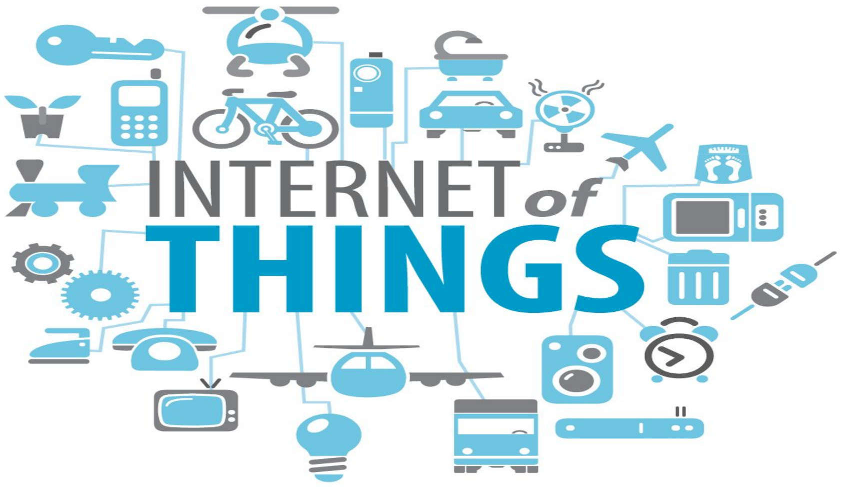
UNIVERSITY OF GRONINGEN
FACULTY OF SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTING SCIENCE

Information Systems

Internet of Things (IoT)



Group 07

AUTHORS
Frans Simanjuntak (S3038971)
Stefan Cretu(S3048438)

December 18 2017

1. Introduction

In this document is presented the solution for the IoT part of the assignment 3 of Information Systems course. The purpose of the assignment is to emulate data collection from an analog sensor using NodeJS. The emulated sensors senses analog values between 0 and 5, being implemented by a logic that generates random numbers within this interval. Then, two different scenarios are considered.

Firstly, every time a new value is sensed, it is checked whether the difference from the previous measured value is bigger than 1. If so, the value is sent to the edge node, in order to simulate data sending from the measuring device.

Secondly, for batches of 10 consecutive sensed values it shall be computed an average, called SMA. This SMA value is sent to the edge node if the difference from the previous computed SMA is at least 0.5. The edge node shall be simulated by using thingspeak.com, which allows for creating a channel used for sending the values and for seeing plots with the received values.

The rest of the document is structured as following: Chapter 2 describes the implementation for both scenarios described above, whereas Chapter 3 discusses the problems with respect to the first scenario by making a comparison to the second one.

2. Solution

In this chapter it is aimed at explaining the implementation of the requirements presented in chapter 1. As already mentioned, the programming language used for this project is NodeJS. Below are explained all implemented functions, one by one.

1. *getRandomArbitrary:*

Firstly, it was implemented a function, called *getRandomArbitrary*, that generates a random number within a given interval. The interval is mentioned by to input values which represent its extremities. Furthermore, generating the random number makes use of the NodeJS library function *Math.random()*. Then, the generated value is returned.

2. *generateRandomNumberForSingleMeasurement*

Secondly, it was implemented a timer event that generates a random value within 0 and 5. It does not take any inputs as it calls the previously defined function to generate a random number which is stored in a local variable. Then, the function *processReadingOnIoTDeviceSingleMeasurement* is called, taking as input the such generated value.

This function is called every 1000 milliseconds, using the NodeJS library function *setInterval,* which takes as input the name of function to be called and the interval for calling, in milliseconds.

3. *processReadingOnIoTDeviceSingleMeasurement:*

It takes as input the random number generated and by the previous function and sends the value to the edge node. It does not return any value, but it just calls the function *sendToEdgeNode.*

However, before sending the value, it filters it by measuring the difference between the newly generated number and the previously generated one. Please note that the previously generated value is stored in a global variable and is initially set to 0, which means that the firstly sensed value is always sent to the edge node. That said, with every new measurement, the difference between these values is computed and stored in a local variable, called *delta.*

Nonetheless, in some cases, a newly measured value can be lower than the previous one, which implies that the such computed *delta* is negative. For these cases, the *delta* is multiplied by -1 as we are always interested in the absolute value of the difference between 2 consecutively sensed data.

Afterwards, the *delta* is compared to 0.1 and everytime it is higher than it, the function *sendToEdgeNode* is called. Moreover, for each value that passes the filtering test, the measurement time is computed and sent as parameter to *sendToEdgeNode*, for obtaining a more realistic and accurate graph on thingspeak.com.

Finally, the global variable storing the previous value is updated to the last sensed value, in order to be used for the next call of this function.

4. *generateRandomBulk:*

This function is called for the second scenario of this assignment. It calls *getRandomArbitrary* to generate a random number between 0 and 5. Thereafter, it calls *processReadingOnIoTDeviceBulk* taking as input the generated value.

This function is called every 1000 milliseconds, using the NodeJS library function *setInterval,* which takes as input the name of function to be called and the interval for calling, in milliseconds.

5. *processReadingOnIoTDeviceBulk :*

This function is called for the second scenario of this assignment. It computes the average of a batch of 10 consecutive generated numbers, called SMA, by cummulating the sum of these values in a local variable and then dividing it by 10. For each value added to the local accumulator, the function is called and computed the SMA when the 10th value is added.

Then, it is computed the absolute difference by comparing this SMA with the previous generated one. If the difference is at least 0.5, the function *sendToEdgeNode* is called and the SMA value is passed as input to it, in order to send the value to the edge node. Also, the date for the SMA that passed the filtering test is also sent as input to *sendToEdgeNode.*

Please note that the previously generated SMA is initially set to 0 and is stored in a global variable. In the end, this global variable is updated every time a new SMA is computed and filtered. Additionally, the accumulator and the index counting the number of values in the accumulator are set to 0 after a new SMA is computed and filtered.

6. *sendToEdgeNode:*

It takes as inputs two values passed from the caller function, that is *processReadingOnIoTDeviceSingleMeasurement* or *processReadingOnIoTDeviceBulk*, depending on the scenario, representing the measured/SMA value and the time of the measurement.

This function handles the communication with thingspeak.com and it emulates sending data to the edge node using a communication channel. For this purpose, firstly it is created a thingspeak client using a client module for accessing the API. The used method, namely *ThingSpeakClient*, is a constructor which requires to specify a server name, that can be localhost or the address of the thingspeak website. Given the purpose of this assignment, the second option was chosen.

Then, a channel is attached to the created client. This operation is mandatory as it is envisaged to write values to this channel and it uses the method *attachChannel,* which requires mentioning an ID for this channel, as well as a write key obtained from thingspeak account.

Lastly, the channel is updated with the value received as input by *sendToEdgeNode*. The method that implements this behavior takes a sinputs the channel ID, the values to be written and an optional callback function, which in this case is used for throwing eventual errors.

3. Discussion

With respect to the first scenario, there were highlighted 2 problems, which are listed and discussed below, in comparison to the second scenario:

1. The first problem is linked to unexpected behaviour in terms of what is monitored.

This problem comes from the physical phenomenon that is measured. Concretely, in reality one or more events influences the values that are sensed. As an effect, during a certain amount of time they sensed values can vary a lot, leading to high differences between 2 consecutively measured values.

Thereafter, this has a direct effect a more frequent call of the data transmission process, in order to send the values to the edge node. Furthermore, similar causes, which are unpredictable in many cases, can lead to low differences between consecutively measured data. However, if the threshold value used to filter the measured data is low, in both cases the data transmission process can be called quite often. That said, when compared to the second scenario, it is clear that the first scenario is more prone to behave in such way.

Nevertheless, this aspect is discussed at the second point.

2. The second problem is linked to energy consumption.

Indeed this is a problem since both sensor reading and data transmission are expensive in terms of processing, which subsequently leads to higher energy consumption.

That said, the first aspect which makes the first scenario to have a higher energy consumption than the second one is the value of the filter (0.1, respectively 0.5). It can be noticed that, in this case, the likelihood for a newly sensed value to be sent to edge node is quite high when compared to the second scenario that uses a filter of a higher value. Therefore, since more values pass this condition in the case of the first scenario, then they are sent to the edge node. So, as more values are sent then the more energy is consumed by the data transmission process.

The other aspect that leads to higher energy consumption is that in the first scenario every sensed value is compared to the given threshold, which makes it a possible candidate for data transmission. Thus, adding the low value for the filter (threshold), there is a high possibility for the data transmission process to be called every 1 second or more than 1 time within a 10 seconds interval. In comparison, the second scenario computes the SMA of batches of 10 consecutively measured values, procedure that contributes to the filtering of data to be sent as, instead of maximum 10 candidate values, only one value is considered for comparison and transmission. That means the data transmission process would be called at maximum once every 10 seconds.

4. Experiment results

In the images below it can be seen the behaviour of both scenarios, observed on thingspeak.com. Firstly, the scenario 1 is illustrated, then the second scenario. In each graph, on Oy axis is represented the measured value and on Ox axis is represented the time (minutes and seconds are written). Please note that thingspeak.com plots data every 15 seconds.

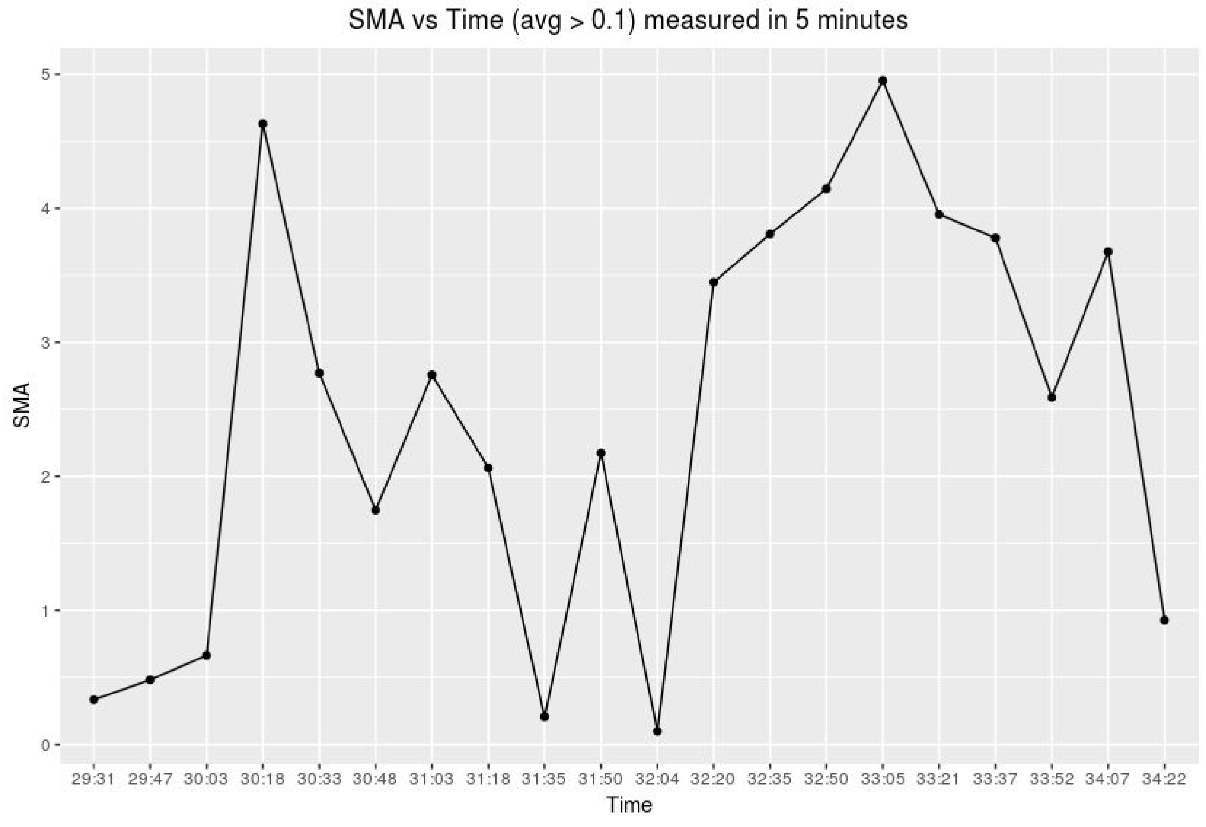


Figure 3.1 First scenario data illustration

Since thingspeak.com plots values every 15 seconds, a slong as they are sent within this interval, in the first illustration it can be seen that a value is drawn every time after 15 seconds. Surely, it does not capture the exact behaviour of the scenario, but we can admit that, if thingspeak could have plotted every 1 second, the intervals between 2 consecutive values would have been lower in the first image. However, it is enough to help us in making some conclusions when comparing to the second scenario.

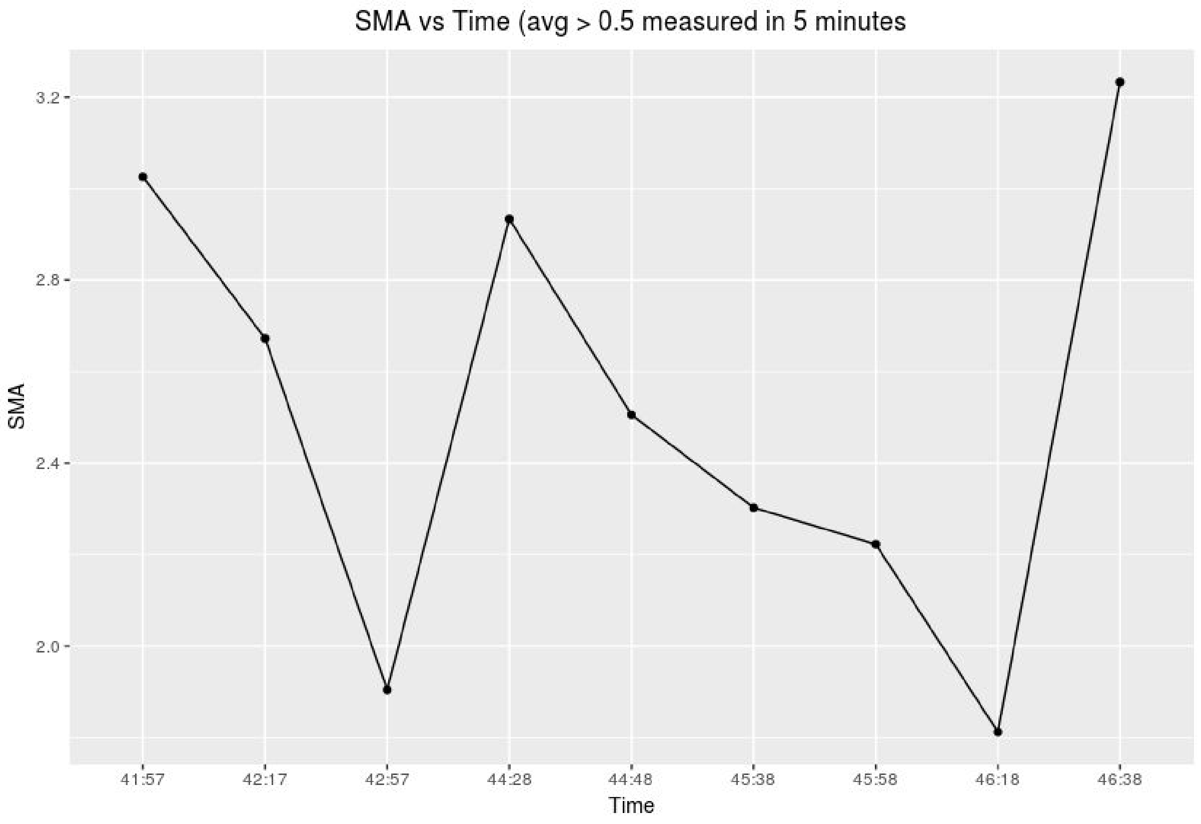


Figure 3.2 Second scenario data illustration

Nevertheless, in the second illustration we can notice that the interval between 2 consecutive plotted values is at least 20 seconds. This result supports the ideas we highlighted above with respect to the better filtering, which is a consequence of using a higher value for the comparison threshold, as well as of applying the idea of computing SMA for 10 consecutive values, that happens every 10 seconds. The results in the above illustration show that, during the experiment, there were never sent 2 consecutively computed SMAs, as there is no interval of 10 seconds.