Open IoT testbed utility cum development framework of virtualized services in terms of sensor as utility

*Kayalvizhi Jayavel, Meenakshi K, Sundara Kanchana J, Srinivas LNB, Revathi Venkataraman
SRM Institute of Science and Technology, Kattankulathur, Chennai, India
*Corresponding Author: kayalvizhi.j@ktr.srmuniv.ac.in

Abstract
Internet of Things aims at connecting everything, everywhere, every time. Most of the industries here predicted billions of connected devices by 2025. Considering the scale, the applications which evolve will also be multifold. This demands huge investments in terms of infrastructure: hardware or software. Taking this into consideration we have proposed an IoT testing cum development architectural framework which will enhance the utilization and reusability factors. The services offered are data, sensor client, actuator client, platform and API. We would like to address this framework as a utility, as users or developers can use our model as "pay as you go model" or demand based model instead of traditional "one pack serves all model". In "One pack serves all" the whole service is provided to users as a monthly or yearly subscription. Traditional frameworks use proprietary devices which create vendor lock in, lack of interoperability and migration issues. We have designed, developed and implemented a prototype with open source boards and tested the reusability metrics in terms of time taken and request-response graphs. We have demonstrated in this paper about sensor data as service, performance enhancement achieved in the database updation and retrieval based on our model.

Keywords: Internet of Things, Testbed, Development framework, Services, Utility, Performance enhancement

I Introduction

Internet of Things (IoT) is a technology capable of equipping the existing things speak to each other anywhere, anytime. IoT-GSI have described IoT as the backbone of this era of information. (International Telecommunication Union, 2017).

The core components of an IoT system are sensors, actuators and the units which process the data and commands respectively. Developers deploy need based applications (S. D. T. Kelly, N. K. Suryadevara and S. C. Mukhopadhyay, Oct. 2013) on the processing units to read the data from sensors or actuate commands to actuators. The processing unit can be as simple as 8-bit microcontroller or as powerful as a processor, which mostly depends on the application in hand. The choice of microcontrollers needs care as they are constrained on energy and size. Many have deployed motes for applications which needs wireless sensor networking. There is possibility to employ open source boards such as the Arduino (Banzi, Massimo, Michael Shiloh, 2014) etc.

This paper uses MQTT protocol as its application layer protocol. The data transfer between parties is taken care by a Protocol Broker. Clients can publish as well as subscribe at various instances of time. The clients will publish on a "topic" to much interested clients will subscribe with. The topics are decided mutually and protocol broker will do the mediation for proper data transfer.

This paper also provides virtualisation of sensors and actuators, thereby enhancing the re-usability of the available infrastructure, and the possibility to access remotely, adds to the re-usability metrics. There may be many reasons as why one may not own a set of devices or needed resources say, cost, malfunctioning, one-time usage, only testing, research purpose or mere experimental data collection, lack of embedded knowledge, lack of breakout boards to support ESP8266 Wi-Fi module (current fluctuations). These issues are mitigated through our service based architectural framework. We have abstracted the complexities to a software developer who will use our utility (Controlled offer of group of services based on user demands) without worrying about these details.

There exist many simulation tools (Fritzing/123dcircuits) as a simple solution for the above-mentioned problem but, they can only mimic, not the real code deployment happens. What if you want to test a real SMS from a GSM module to be sent. Thus, our paper provides a development and testing utility which offers sensor data, actuator, platform and API as services. Our major highlights are zero learning curve for users/developers, socially reusable with zero development cost and 24x7 availability, teaching/testing/experimental platform usable by academicians, industrialists or analysts.

This paper is formulated with section II comprising of state of the art. Section III is our proposed architecture. Section IV is our implementation and results. Section V is conclusion and Results.

II. State of the art

Our research is to design, develop, deploy an open IoT test bed cum development utility which offers sensor data, actuators, platforms and API as service. This paper considers sensor data as service in detail. The other services like actuators, platforms and API's are covered in our next paper. Thus, our research requires literature review at two levels. There is a need to understand the existing test bed frameworks, their purpose, the services, challenges or issues. Secondly to investigate the way each service offered, their methodology, merits issues or challenges. Thus, we have formulated our literature survey in two genres, Test bed and Service respectively.

(De, P., Raniwala, A., Sharma, S., & Chiueh, T. C., 2005)The word testbed was almost synonymous to Wireless Sensor Network Testbed and most of the boards used are proprietary which lead to interoperability problems. The main aim of these testbeds is to check how various network level protocols and applications perform with non-IP based sensors networks and was not developed with IoT in mind. Our initial thought process was to rectify the issues, but later understood that it is almost obsolete to think of Sensor networks in terms of motes, a new paradigm shift of open source everywhere made us rethink and develop our architecture.

(Zorzi, M., Gluhak, A., Lange, S., & Bassi, A. , 2010)The authors of this paper have attempted to provide an architectural framework to overcome the current fragmentation and limitation of solutions, where many “Intranets” of Things exist, towards a true “Internet” of Things, where all devices will be part of a globally integrated system. This paper realized it early we should say, the need for a unified approach for IoT when many of them are still busy developing single applications in silos. They have admitted the agony of only small group of enthusiasts from academia, industry and public institutions working to bring up a unified model for IoT. They have emphasized the dire need for standardization and ETSI have made some notable contributions.

(Clement Burin Des Rosiers, 2011) have developed a Sense Lab- a very large scale opens Wireless Sensor Network Testbed. This testbed is a generic testbed which allows experimental research of protocols used for communication and algorithms at application level. The important feature of this testbed is, the algorithms under test need not be of specific domain or category. There is scope for improvement on concurrency and heterogeneity aspects of an IoT experimentation set up.

(Coulson, et al., 2012) WISEBED was initially started in 2010 later evolved to include concepts of virtualization to the existing testbed architecture. It uses a generic XML-based language (WiseML) to describe about the experimental, in setting up and for result storage. The events can be booked using database backed google calendar or in-memory storage. This offers access to on-going experiments through web interface via web services to adjust parameters, along with monitoring and collection of data. They have proposed virtual testbeds and their integration with physical testbeds. There is scope for improvement on heterogeneity and node level virtualization as against testbed level virtualization (resolve access to same hardware resources) aspects of an IoT experimentation set up.

Many more community oriented platforms like Sense Web (Grosky, William I., Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao, 2007), Global Sensor Network (Aberer, Karl, Manfred Hauswirth, and Ali Salehi, 2007), Sensor Base (Chang, Kevin, Nathan Yau, Mark Hansen, and Deborah Estrin, 2006)], Iris Net (Gibbons, Phillip B., Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan, 2003) and Semantic Sensor Web (Sheth, Amit, Cory Henson, and Satya S. Sahoo, 2008) have been established which allowed users to share the data from heterogeneous data sources.

SenseWeb, an Microsoft's creation (Grosky, William I., Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao, 2007) provides a generic platform to share, query and visualize sensor data. SenseWeb provides various tools for data owners and data users to publish and subscribe the data respectively. SensorMap is a geographical web interface provided by SenseWeb to query the needed data and get the visualization of the same.

Global Sensor Network (GSN) (Aberer, Karl, Manfred Hauswirth, and Ali Salehi, 2007) offers a general-purpose infrastructure which can be programmed based on user needs as against usual collection only model from a central repository. The GSN middleware infrastructure attempts to address heterogeneity by integrating heterogeneous sensor networks and this is achieved by deploying the GSN middleware on any computer which is interested in interacting with one or more sensor networks.

IrisNet (Gibbons, Phillip B., Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan, 2003) (Internet-scale Resource-Intensive Sensor Network Services) aims at providing a sensor web which can be accessed from anywhere, anytime fulfilling the requirement of an IoT based system. This attempt provided multitude of sensors openly accessible to users from all walks of life.

Thus, our work attempts to address all the issues, as to reduce the learning curve, the data analysts must undergo, to provide categorization based on sensors granulated at parameter, better visualization, and removing the dependency of any coding skills or hardware knowledge required to create the application that generates the data. Hence attempting to consolidate, we investigated testbed papers of varied goals be it educational or industrial or testing or development, all of them had one issue in common, they all used proprietary hardware or software or both. This left the major challenge of interoperability unattended. Thus, our work attempts to provide testbed as a utility with proof of concept to demonstrate that use of open source boards and technologies comfortably takes care of the prime requirements of IoT (as stated by many authors) like heterogeneity, interoperability and reusability appreciably well.

III. Proposed Architecture

We have proposed an IoT architectural framework and Infrastructural prototype which is shown in **Figure1a and Figure 2a. Figure 2a** explains the infrastructal details comprising of hardware and sofware deployments to achieve the utility based IoT framework.

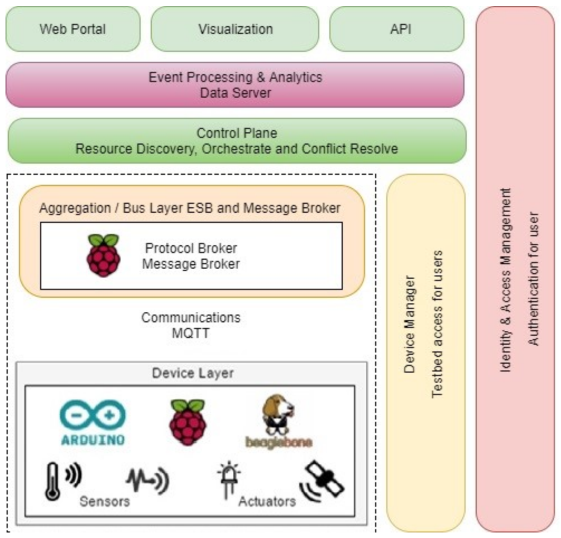
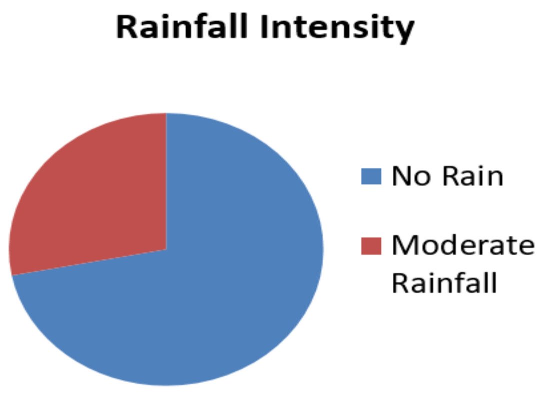


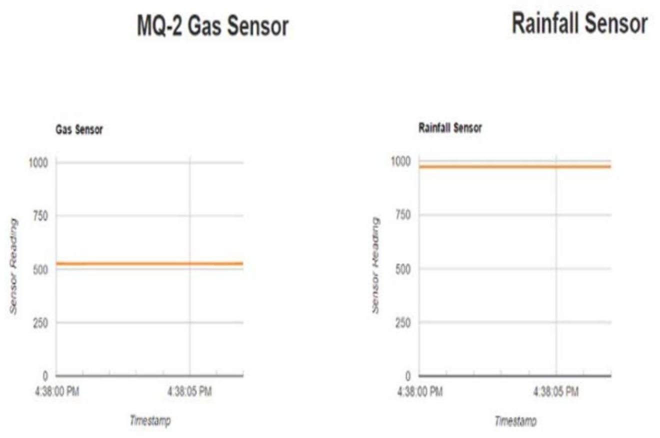
Figure 1a: IoT architectural framework as utility

The architecture comprises of Device layer, Communication layer, Aggregation layer, Control plane, Event Processing and Analytics, Web portal, Visulaization, API, Device manager, Identity and

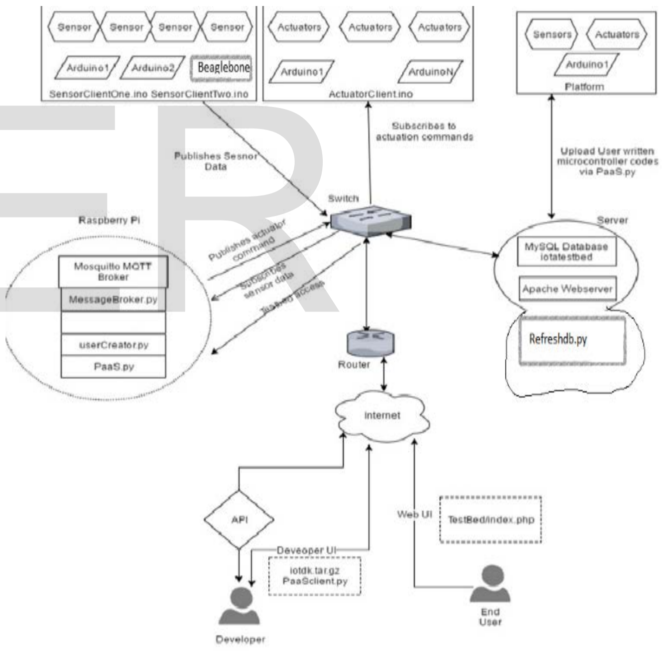
Access management. We have developed a prototype which can be used for testing and development purpose. Device layer comprises of open source boards and sensors and actuators connected to it. Comminication layer is useful in communicating the data collected from underlying sensors and command to actuators from users using any device to server protocol. Some examples are MQTT (Message Queueing Telemetry Transport), CoAP (Constrained Application Protocol) and XMPP (Extensible Messaging and Presence Protocol). We have deployed MQTT for its visible benefits towards event driven scenarios. Aggregation layer is used to pack the data from various senor clients as need or application demands. This will help in scenarios to avoid redundant data, remove erroneous data, avoid sending unchanged data and many more. MQTT protocol broker takes the resposibility as aggregator. The data received from sensors are updated to the database via message broker script. The analyst can use data available in the database server (Maria DB) and as the event processing depends on the application in hand, is left to the user or developer to handle. The visualization **(Figure 1b-1c)**is provided by our Dashboard via our website. Web portal is our website which accepts requests from local or remote. API management helps in downloading the APIs for developers to actuate commands or utilize as



libraries in their code development. Device management is responsible for monitoring the underlying hardware including sensors. Platform as service utilizes the service of the device manager.



Identity and access management controls and authenticates right users and developers to utilize our utility based service. We have authentication at



two levels: Web portal at local access and session login in remote SSH.We have designed and successfully integrated all of the above to provide utility based service to users and developers.

Figure 1b: Rainfall Intensity Dashboard visualization

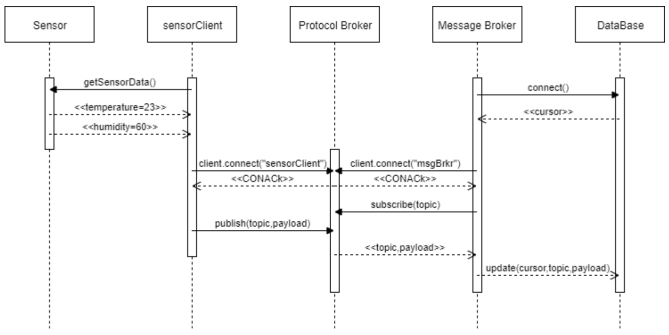
Figure 1c Gas and Rainfall sensors Dashboard Visualization

Figure 2a: Infrastructural IoT framework as utility

Our utility framework is designed with goal of offering services at data plane, hardware plane and software plane. Sensor data is offered as service at the data plane. Our framework is capable of providing data in standard data exchange formats say JSON, XML and CSV. At the hardware level services can be catagorized as sensor client and actuator client. The software plane services are offered at two dimensions namely remote code porting and API as service.

The requests from user and developer reaches our gateway (Pi). The gateway has Mosquitto protocol broker, a message broker, user create script, platform as service (paas acript). All the sensor and actuator clients register to the protocol broker based on topics provided by the protocol broker. Sensor clients publish data on the topic and protocol broker subcribes on that topic. In case of actuator client, protocol broker publishes commands on the respective topic and actuator client subscribes on the same topic. Message broker script is developed to update the data to the database for data analysts to work up on or to create legacy database for later point of usage. Usercreator script is the login and authentication module. Paas script enables coder to use shell remotely. We have deployed MySQL database server and Apache web server on the server side to enable data storage and remote access. Figure 2b summarises the complete data flow in a nutshell.

Figure 2b: Sequence diagram depicting overall communication.



IV. Implementation and Results

We have implemented our architectural framework to provide proof of concept. We have developed our prototype with open source boards. We have incorporated sensors namely DHT11, MQ-2, MQ-7, BMP-180, GPS, GSM, Buzzer, LEDs, RGBs, Color sensors, LM35, Potentiometer and LCD. The sensors are connected to Sensor Client 1 and Sensor Client 2. Actuators are connected to Actuator client and platform as service with sensors and actuators. The framework provides features for users and developers to exploit, according to their skill set and expertise.

Sensor data as service

Based on the literature survey, one can classify sensor data collected from sensors after reaching the message broker via the protocol broker, should reach the database. We have deployed Maria DB as our database server. The idea behind the classification is as follows. If we consider a scenario with say two sensor clients: sensor client 1 and sensor client 2. Each sensor client has variety of sensors say, DHT11, MQ2, Colour sensor and BMP-180. Except DHT11 every other sensor sends only one parameter as its payload value. But there are many sensors like DHT11 which can measure multiple parameters of the environment. Having said this, researchers have sent the data in three styles: all the sensor readings of a sensor client as a single payload and updated in single table, all the sensor readings on change of a sensor client as a single payload and updated in single table, each sensor readings on change as separate payload of a sensor client and updated in single table. And our model is to send each sensor readings on change as separate payload of a sensor client and updated in separate tables. We would like to categorize the updation models in to 4 broad groups listed below and we have compared the pros and cons with real data sources. We have found based on our experimental model scenario 4 (Respective Topic on Change Respective Table) performed better, hence adopted in our model. Table 1 provides the sensor data from various sensors and Table 2 the analysis chart.

Scenario 1: Single Topic Single Table
Scenario 2: Single Topic on Change Single Table
Scenario 3: Respective Topic on Change Single Table
Scenario 4: Respective Topic on Change Respective Table

Approach 1:	
$N_r = N_m$	11
Approach 2:	
$N_r = N_m = N_c$	04
Approach 3:	
$N_m = N_{ci}; N_r = N_c$	02, 04
Approach 4:	
$N_m = N_{ci}; N_r = N_{ci}$	02, 02

Table 1: Sensor data from various sensors

Let

$N_s$sensors with N_t topics

N_c denote change in topic

N_{ci} denote change in respective sensor

N_r denote number of records

N_m denote number of messages

Reduce N_r and N_m as much as possible especially for sensors whose readings hardly change.

Reading Number	Sensor 1	Sensor 2	Sensor 3
1	10	15	30
2	10	15	30
3	10	15	30
4	10	15	31
5	10	15	31
6	10	15	31
7	10	15	31
8	10	15	31
9	10	16	31
10	10	16	31
11	11	17	32

Table 2: Analysis data chart

Thus, the above statistics proves that approach 4 with scenario 4 shows the best results compared among the other scenarios. Let us see in detail how this per topic per parameter model works better than per topic per client model.

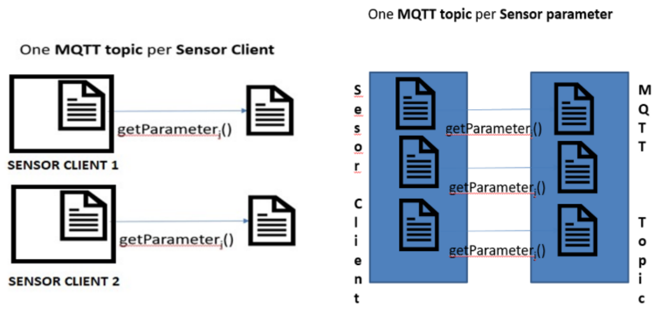


Figure 3a and 3b: Comparison between one MQTT topic per client vs per parameter

Figure 3a shows the visual representation of one MQTT topic per sensor client model. **Figure 3b** shows the visual representation of one MQTT topic per parameter. In our architecture, the sensor readings published to the protocol broker (Mosquito in our Pi) will be published again to the message broker (python script in our Pi) and the same data after pre-processing to avoid error data and null values will be sent with recent time stamp to our database server. The first half of data transfer between clients and protocol broker will not have great impact because of the following reason. Theoretically, only few milliseconds would be taken for concatenating the strings, which will be compensated by reducing number of publish. And, also if any sensor reading was incorrect, then the whole cycle will be re-done which happens at very low rate. The actual saving in adopting approach 4 will be visible only in the second half of data transfer between message broker and database. In message broker, parsing the string into different values and assigning to respective variables takes up time. Moreover getParameter() (one of our API) will return same payload, which creates excessive unused payload. For instance, consider a sensor client with DHT11 and Gas, parsing should happen at 2 levels: Parse at sensor level (DHT11, Gas), Parse at parameter level (Temp, Hum). And the update time for all the sensors will be same and there won't be individual graphs for each sensor.

Polling is used as an alternative to all the above. All the parameters are collected and appended in a single message. And a unique function which uses global variable for each function to collect the values. A script runs continuously as a back-ground process and keeps collecting data irrespective of call being initiated and updates the global variable for each parameter. This obviously consumes excessive processing cycles.

Both MQTT per client and the polling style are poor at error handling. The error may be due to the sensor is faulty or damaged. This disrupts other sensor data flow. In the former, the valid data needs to wait for error rectification which is a huge tradeoff or send with placeholders which incurs excessive payload. The parsing will be done only to find the error data and re-subscribe to sensor client again. If any sensor reading was incorrect or error, then the whole cycle will redone (happens at very low rate). But if this situation occurs, one whole cycle will be skipped. And rate of sensor data receiving will be reduced by 1 at that particular instant alone. Additional time consumption occurs when the program requires actual value to run, so it should wait for next set of values.

But in our one, MQTT topic per Sensor parameter there is no excessive unused payload, no excessive computational cycles as individual parameters have individual MQTT topics. And moreover getParameter$_j$() function would have to subscribe to the topic only once, get the payload and return it without any parsing. Figure 4a shows the updation time via one MQTT per parameter takes less time to update when compared to when sent all in one (800msec). Figure 4b shows the database updation time for Gas and Rainfall sensors.

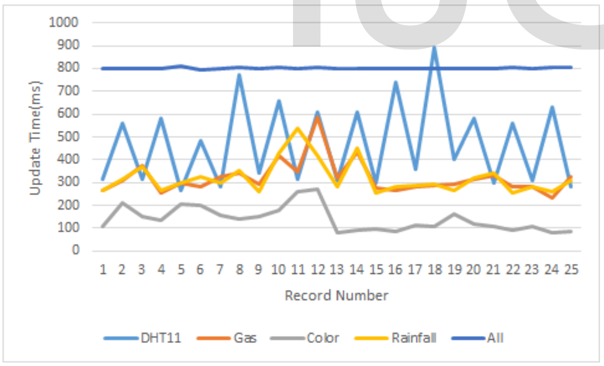


Figure 4a: Comparison on updation time: Approach 4 outperforms

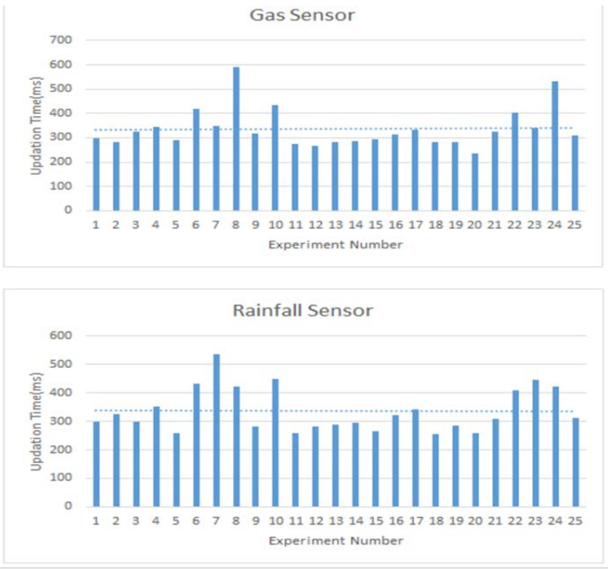


Figure 4b: Sensor database updation time

V. Conclusion and Future work

We have designed, developed and implemented our IoT architectural framework through our open source infrastructural set up. The services offered are collectively qualified as utility as we are offering it as on-demand or use as you want. Necessary graphs are provided to prove improvement at all levels. Thus our utility based IoT model have performed well in terms of request-response ratio, database update and retrievel interval and utilization factor. We have demonstrated our model using Message Queueing Telemetry Transport and we are attempting to develop a similar model using MQTT-SN. We would like to add security aspect to our model and demonstrate this architecture is fool-proof as well.

References

Aberer, Karl, Manfred Hauswirth, and Ali Salehi. (2007). Infrastructure for data processing in large-scale interconnected sensor networks. *2007 International Conference on Mobile Data Management* (pp. 198-205). IEEE.

Banzi, Massimo, Michael Shiloh. (2014). *Getting Started with Arduino: The Open Source Electronics Prototyping Platform.* . Maker Media, Inc., .

Chang, Kevin, Nathan Yau, Mark Hansen, and Deborah Estrin. (2006). *Sensorbase.org-a centralized repository to slog sensor network data.* Los Angels: National Science Foundation.

Clement Burin Des Rosiers, G. C. (2011). SensLAB Very Large Scale Open Wireless Sensor Network Testbed. *Proc. 7th International ICST Conference on Testbeds and Research Infrastructures for the Development.* Shanghai, China: The open archive HAL. Retrieved from sensLAB: https://hal.inria.fr/inria-00587862

Coulson, G., Porter, B., Chatzigiannakis, I., Koninis, C., Fischer, S., Pfisterer, D., . . . Baumgartner. (2012). Flexible experimentation in wireless sensor networks. *Communications of the ACM, 55*, 82-90.

De, P., Raniwala, A., Sharma, S., & Chiueh, T. C. (2005). Design considerations for a multihop wireless network testbed. *IEEE Communications Magazine, 43*(10), 102-109.

FIT consortium. (2014). *IoT-Lab*. Retrieved from FIT IoT-Lab: https://www.iot-lab.info/what-is-iot-lab/

Gibbons, Phillip B., Brad Karp, Yan Ke, Suman Nath, and Srinivasan Seshan. (2003). Irisnet: An architecture for a worldwide sensor web. *IEEE pervasive computing* (pp. 22-33). IEEE.

Grosky, William I., Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. (2007). Senseweb: An infrastructure for shared sensing. *IEEE multimedia*.

International Telecommunication Union. (2017, August 16). *ITU* . Retrieved from JCA-IoT and SC&C: http://www.itu.int/en/itu-t/jca/iot/Pages/default.aspx

Sheth, Amit, Cory Henson, and Satya S. Sahoo. (2008). Semantic sensor web. *IEEE Internet computing*. IEEE.

S. D. T. Kelly, N. K. Suryadevara and S. C. Mukhopadhyay. (Oct. 2013). Towards the Implementation of IoT for Environmental Condition Monitoring in Homes. IEEE Sensors Journal , 13 (10), pp. 3846-3853.

Zorzi, M., Gluhak, A., Lange, S., & Bassi, A. . (2010). From today's intranet of things to a future internet of things: a wireless-and mobility-related view. *IEEE Wireless Communications, 17*(6).