Sensorian Hub: An IFTTT-based Platform for Collecting and Processing Sensor Data

Dylan Kauling and Qusay H. Mahmoud
Department of Electrical, Computer and Software Engineering
University of Ontario Institute of Technology
Oshawa, ON, L1H 7K4 Canada

Abstract—**The Sensorian Hub is a software platform designed and developed for using the Raspberry Pi Single Board Computer (SBC) as a sensor node, whether independently or in a larger network of sensors. The intention of the platform is to provide an easy to use and highly customizable system for collecting and processing sensor data in a variety of environments and applications. In this paper, the design and implementation of the system through the use of a Python client, a PHP and Flask Web Server/API, a Relational Database Management System (RDBMS) MySQL and external connections to services such as If This Then That (IFTTT) are discussed in detail. Some sample applications and a prototype implementation with the Sensorian Shield are presented as well to demonstrate the current capabilities as well as the opportunities for future work.**

Keywords—CoAP, Data systems, Home automation, Internet of Things, M2M Communication, Raspberry Pi, REST, Sensors.

I. Introduction

Setting up a sensor network for monitoring anything from temperature to humidity in a given environment can be both costly and time-consuming. Commercial sensor nodes, while typically well suited to the task, can often be prohibitively expensive and require specialized knowledge in their operation. An alternative for simpler implementation people have been flocking to are the use of multi-purpose single board computers (SBC) such as the Raspberry Pi (RPi) to rapidly prototype and develop their own solution. The RPi is a credit card sized SBC that is designed to promote the learning of computer programming at younger ages in schools [1]. The Pi is relatively inexpensive and easy to obtain and also offers the ability to easily enhance their capabilities through the use of shields and breakouts. These are versatile, configurable accessories offered by a wide variety of providers. Each shield is designed with the purpose of adding certain functionalities to the Raspberry Pi. One such shield is the Sensorian [2] and is where both the title and motivation for this project is derived. Within the board, it houses multiple sensors such as an ambient light sensor, barometer, altimeter, pressure sensor, and 3 touch sensitive buttons [3]. This gives the user a platform that is capable of providing them with a wide range of continuous real-time data. This is only one possible solution for extracting data from the world around the sensor, with the intention of supporting a wide variety of sensors already in use today. In our project we made it easier to set up the Sensorian shield, as well as provided users with an easy way to view the data along with a visual representation of the data that is being returned.

With such a wide variety of options out there for sensors, let alone ways to implement them, there can be a lot of unnecessary redundancy in development effort. We aim to provide a flexible solution that will enable the average user to harness these technologies to perform desirable tasks. When looking at the current mainstream alternatives, such as sensors and commercial Internet of Things (IoT) devices, they are almost always proprietary hardware coupled with limited and closed source software. This means their usefulness is dependent on what the developer provides in terms of functionality and that they may not work on all platforms. However, with the Sensorian Hub, we aim to make it platform agnostic, working with multiple languages by using a REST API to allow all of the sensors to interface. Finally, by making these kits easier to set up, they will have a smaller learning curve and be more attractive and customizable than other standard offerings.

Within this project we sought to create a flexible system where the Sensorian Shield would be networked through a client-server based architecture. This would facilitate the acquisition and processing of sensor data. Moreover, in keeping with the spirit of the Sensorian project and the intention of supporting a wide variety of sensors, quite possibility through assistance from others, we have made all of our code fully open source on GitHub. This will enable the community to repurpose it however they deem fit and also enable them contribute back and assist us with future development of the project. We hope this makes it easier and makes the platform more feature-rich, eventually fostering a large community of users who too would be interested in contributing to the project.

The following sections describe related work that we wished to improve upon, the project's capabilities in its current state, and plans for how to improve functionality in the future.

II. RELATED WORK

The use of the Raspberry Pi as a Sensor Web node for home automation has been explored in [11]. Using a similar strategy for displaying the data through a RESTful service and Apache, our implementation differs in that it is designed to be hosted separately from the Pi and to store the data from as many sensor nodes as desired. While the server could also be hosted on one of these nodes, having it hosted in a central location and with unrestricted access to the Internet gives the added benefit of being able to interface with other useful services such as If This Then That (IFTTT) [14]. Also, while the paper describes the use of the sensor node in a web, there is no indication as to how this is achieved. Our solution utilizes a Relational Database Management System (RDBMS), in this case MySQL, to aggregate all the data and group it together based on users and hardware unique IDs.

Another example of using the Pi as a tool for home automation is discussed in [10]. This is strictly a command and control implementation for automation, without any sensor inputs or logic performed locally on the device. The authors explore the use of email to control a device in the home remotely based on the message content. Our solution differs in that email is just one of many possible control methods after being integrated through the IFTTT Maker channel. Besides the use of RESTful operations locally or remotely through the use of a relay server, any other possible triggers from channels on IFTTT could be used to invoke a request to our Raspberry Pi. This is also more performant and reliable as emails can often unpredictably take much longer to arrive than normal.

As for more sensor-based applications, another group of researchers utilized a Raspberry Pi to collect medical data [4]. This was achieved through the use of an accelerometer attached to the Pi along with a Bluetooth module to connect with a phone to receive the data. A significant shortcoming of this approach is that the data cannot be provided in real-time to the client, it must be requested, transferred and decrypted. The primary benefit of note of their solution is that of the encrypted transport of the data. While encryption is not currently a feature of our prototype, being built on a REST framework would allow for easy adoption of SSL and certificate-based encryption in addition to the Basic HTTP Authentication already in place through the use of Flask-HTTPAuth [18].

Lastly, another interesting system design using Raspberry Pi and Arduinos to collect sensor data is explained in [5]. This implementation differs in the way the data is communicated by sensors nodes and the server. Instead of USB WiFi adapters, the researchers chose ZigBee modules, which allows for mesh networking without a pre-existing 802.11 wireless router. This has the benefit of compartmentalizing their solution to its own network, with only the base station/coordinator communicating with the Internet and being able to scale out the network with more sensor nodes added to the mesh. However, this has the limitation of requiring more nodes to specifically act as routers between the sensor nodes and the base station, increasing the network complexity. While this may be a viable solution in some locations, the ubiquity of the wireless 802.11 standard has seen access points installed into most buildings in a multitude of places to ensure full coverage, making them a good option for most projects short of an extremely large scale.

In summary, the Sensorian Hub differs from other solutions in a number of ways. The Hub functions exactly as the name would suggest, in that it is designed to aggregate data from numerous sources in a central location, in contrast to other solutions which only collect from a single source. Instead of being restricted to only email for external communication and automation, IFTTT and RESTful support allow for a much greater amount of options that are also faster and more reliable. After setup, the entire data collection process is completely autonomous and does not require the manual uploading of data files like in other solutions. The Sensorian Hub is also based off well-defined and ubiquitous standards such as HTTP and 802.11 which makes it easy to integrate into current networks and systems. Finally, and most significantly, the Sensorian Hub is a fully open-source solution, giving the user everything they need to get started the freedom to modify it however they deem

III. Sensorian Hub

The proposed solution to the lack of an easy, all-encompassing client for use of a Raspberry Pi as a sensor node was the creation of the Sensorian Hub client and its partner web application. The main goal of this solution was to create a basic, modular code base for getting the average user off the ground easily with their sensor-based application and started with deeper integration of its functionality.

As shown in Fig. 1 the Sensorian Hub is broken into three main parts. The Web Server hosts the site which receives the sensor data from the Clients, stores and retrieves this data from a MySQL Database for display on the site to users, and accepts registrations of new users and sensors. The public-facing Relay allows for communication with Clients behind secure networks by listening for connections established by the Clients first. This allows for commands to be sent from users outside the local network or other cloud services such as IFTTT.

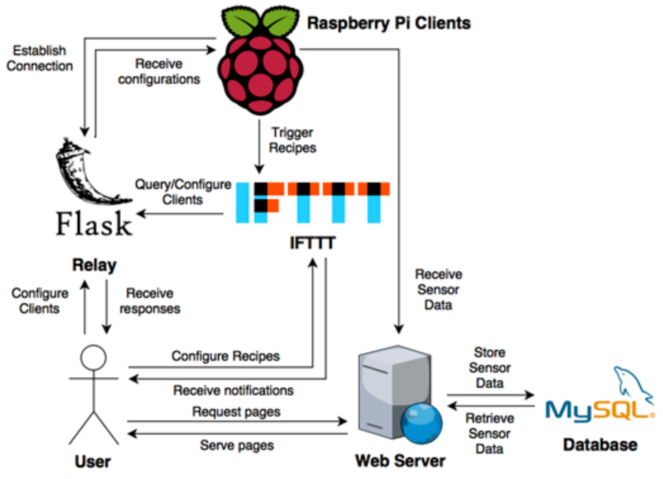


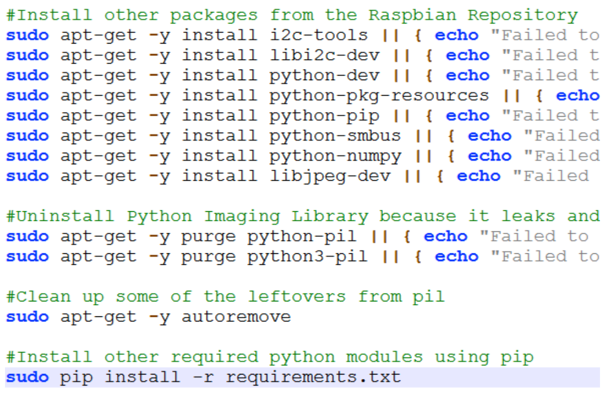
Fig. 1. High-level Sensorian Hub overview.

The following subsections will go over the pillars of our design philosophy and describe how we sought to achieve each of these objectives.

A. *Ease of Use*

As previously stated, one of the main problems we wanted to solve was that of the requirement for in-depth knowledge of the particular sensor being used. One of the main barriers to entry with any new product is the learning curve associated with using said product. While the maker community, that is to say those who enjoy tinkering and technology-based do-it-yourself projects, are generally a savvy bunch, there are certainly those that are much more proficient with a soldering iron than a command terminal. It is this distinction that led us to create a program that could be easily installed, configured, used and modified no matter the user's technical experience and the needs of their project.

This philosophy had to permeate every aspect of the design of the client. The first and most basic manifestation of this was simply commenting the code and having logical, human-readable variable and function names. Sticking to these best practices makes it much easier for an onlooker to understand foreign code. These comments were complemented by a basic readme file on code repository as to how to get up and running with the client. One notable part of the readme was the instructions for the installation script which was included with the code.


```
#Install other packages from the Raspbian Repository
sudo apt-get -y install i2c-tools || { echo "Failed to
sudo apt-get -y install libi2c-dev || { echo "Failed t
sudo apt-get -y install python-dev || { echo "Failed t
sudo apt-get -y install python-pkg-resources || { echo
sudo apt-get -y install python-pip || { echo "Failed t
sudo apt-get -y install python-smbus || { echo "Failed
sudo apt-get -y install python-numpy || { echo "Failed
sudo apt-get -y install libjpeg-dev || { echo "Failed

#Uninstall Python Imaging Library because it leaks and
sudo apt-get -y purge python-pil || { echo "Failed to
sudo apt-get -y purge python3-pil || { echo "Failed to

#Clean up some of the leftovers from pil
sudo apt-get -y autoremove

#Install other required python modules using pip
sudo pip install -r requirements.txt
```

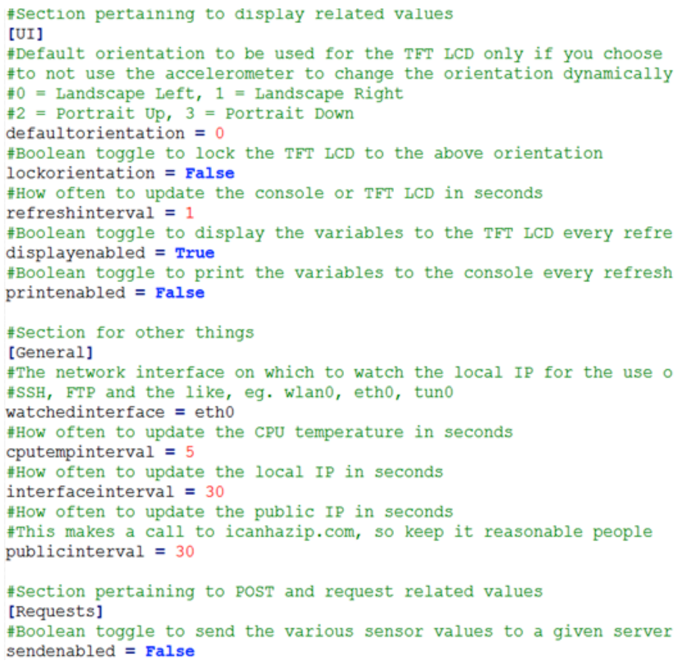
Fig. 2. Install script for Sensorian Hub Raspberry Pi client.

This script as shown in Fig. 2 was another addition to facilitate the easiest and most pain-free experience possible for a new or even existing user. The script automatically installs the required Python libraries and builds the included C DLLs required by the client. This can take a fresh image of Raspbian, the most popular Linux distribution for the Raspberry Pi, and prepare it for use with the client. Lastly, the included configuration file was created to make it easy for the user to change necessary and arbitrary values alike, without having to dig around in the source code to do so. The use of this configuration file will be further described in the following subsection.

B. Modularity/Customization

Since this client is meant to be used as the basis for other projects, another pillar of our design philosophy was that of modularity and customization. Since not every user will have the same sensors and may not require every function of the client or with the same parameters, we wanted to make as much of this easily configurable as possible. The two fundamental components for this to be possible within our client were the configuration file and threading implementation.

The configuration file as briefly described in the previous section allows the user to tailor the client to their specific needs through the toggling of specific functionality or tweaking of certain parameters. This is a simple file with the various values the user can alter separate from the code that is parsed on the program's startup. Should any of the values be missing or even the file itself, it gets written back to disk with some defaults set in the code. In line with previous tenant encouraging ease of use, a sample configuration file is included in the code repository that clearly defines what each variable does and gives some expected values. Having an example file as shown in Fig. 3 has the added benefit of preventing the user's settings from being overridden if they performed a git pull and protects the comments from being removed when the program rewrites the configuration file.


```
#Section pertaining to display related values
[UI]
#Default orientation to be used for the TFT LCD only if you choose
#to not use the accelerometer to change the orientation dynamically
#0 = Landscape Left, 1 = Landscape Right
#2 = Portrait Up, 3 = Portrait Down
defaultorientation = 0
#Boolean toggle to lock the TFT LCD to the above orientation
lockorientation = False
#How often to update the console or TFT LCD in seconds
refreshinterval = 1
#Boolean toggle to display the variables to the TFT LCD every refre
displayenabled = True
#Boolean toggle to print the variables to the console every refresh
printenabled = False

#Section for other things
[General]
#The network interface on which to watch the local IP for the use o
#SSH, FTP and the like, eg. wlan0, eth0, tun0
watchedinterface = eth0
#How often to update the CPU temperature in seconds
cputempinterval = 5
#How often to update the local IP in seconds
interfaceinterval = 30
#How often to update the public IP in seconds
#This makes a call to icanhazip.com, so keep it reasonable people
publicinterval = 30

#Section pertaining to POST and request related values
[Requests]
#Boolean toggle to send the various sensor values to a given server
sendenabled = False
```

Fig. 3. Sensorian Hub RPi client configuration file.

The configuration file splits the values into sections related to the different types of sensors supported and some other functionality such as IP address monitoring and LCD output. The most notable values are those to configure the polling rate of the various sensors. Not only can these values be changed in the configuration file directly to be loaded up the next time the program is run, but they can be changed either locally or remotely as well. With physical access to the device and LCD screen, one can press a button to bring up a menu, shown in Fig. 4, through which they can set many of the basic options found in the configuration file to some common values.

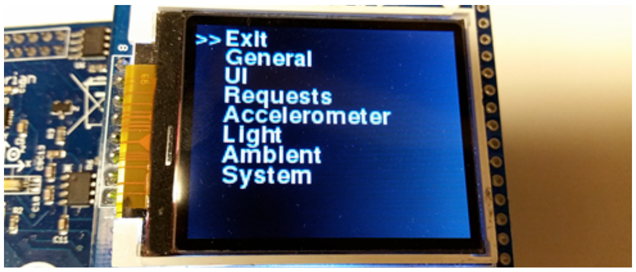


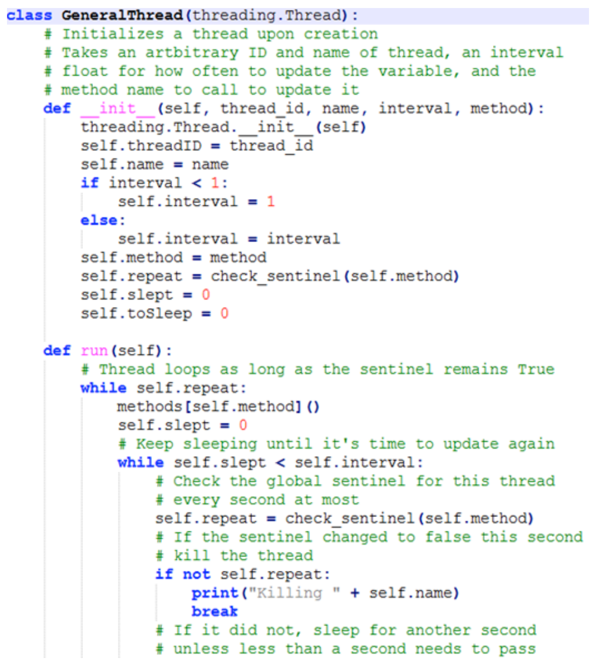
Fig. 4. Sensorian Hub RPi client configuration menu.

For more fine-grained control, the client also optionally hosts a Flask RESTful [17] API to receive new configuration settings which are applied immediately, in addition to commands to reboot, shutdown and kill the program. During development, the Constrained Application Protocol (CoAP) [6] was also explored but, in keeping with the previous tenants, a fully HTTP REST implementation was deemed simpler and more adaptable to existing projects.

In order to poll the different sensors at multiple different and arbitrary rates we had to implement multithreading. The use of threading in the client allows each sensor/function of the client to be executed at a different rate or even potentially not at all should it not be required. Each function and sensor is given its own thread with a general thread class, shown in Fig. 5, that calls the function to update its related variables at the rate specified in the configuration file or program defaults.

C. Robustness

Pursuant to the always-on nature of Internet of Things and sensor-based devices, the last primary pillar of design for the client was robustness, including speed, functionality and first and foremost, reliability. The Raspberry Pi already has a proven track record of being a reliable, performant, always-on device, capable of a multitude of IoT-related tasks with minimal error [7]. Building upon these capabilities we sought to ensure that, despite everything we intended to implement in the client with all the supported sensors and other system functions, we did not introduce any additional sources of system instability in our code. To ensure that this was the case, we performed various tests on the code which will be outlined in a following section.



```
class GeneralThread(threading.Thread):
    # Initializes a thread upon creation
    # Takes an artbitrary ID and name of thread, an interval
    # float for how often to update the variable, and the
    # method name to call to update it
    def __init__(self, thread_id, name, interval, method):
        threading.Thread.__init__(self)
        self.threadID = thread_id
        self.name = name
        if interval < 1:
            self.interval = 1
        else:
            self.interval = interval
        self.method = method
        self.repeat = check_sentinel(self.method)
        self.slept = 0
        self.toSleep = 0

    def run(self):
        # Thread loops as long as the sentinel remains True
        while self.repeat:
            methods[self.method]()
            self.slept = 0
            # Keep sleeping until it's time to update again
            while self.slept < self.interval:
                # Check the global sentinel for this thread
                # every second at most
                self.repeat = check_sentinel(self.method)
                # If the sentinel changed to false this second
                # kill the thread
                if not self.repeat:
                    print("Killing " + self.name)
                    break
                # If it did not, sleep for another second
                # unless less than a second needs to pass
```

Fig. 5. Sensor update polling thread code.

IV. PROTOTYPE

The above design philosophies culminated in what became two main parts of our solution, the client [8] and the website [9]. For the most versatile demonstration for our prototype, we chose to write our client to support the Sensorian Shield first. The client has already been somewhat described above. It is a Python program for the Raspbian OS on Raspberry Pi which depends on the Sensorian firmware and hardware to collect and send sensor data to a given server. Said server collects this data and displays it in a number of formats useful to the user. A more in-depth explanation of the website process is as follows.

Fig. 6 illustrates how the individual components of both the server and client systems interact with each other and the world. As previously described, the code is broken into two main components, the client and the server, which run on the Raspberry Pi with Raspbian and a Linux or Windows AMP stack respectively. The Raspberry Pi then interfaces with the sensors, or in the case of our prototype the Sensorian Shield, through the use of the General Purpose Input Output (GPIO) pins on the Pi. As the client collects data, it optionally sends it to the database server through the use of the Python Requests library [15], to be received by the PHP-based RESTful API of the server. Alternatively, this data can also be sent to other services such as IFTTT through their Maker channel using the same Requests library. Both the server and the client also host Flask [16] servers to handle incoming HTTP requests intended for reconfiguring or sending commands to the clients remotely. If the user is on the local network or has remote access to the client through port forwarding on their router, they can send HTTP requests directly to the client to command or reconfigure the device.

If the user does not have direct access or wishes to use another service such as IFTTT's Maker channel to communicate with the Pi, the requests can be sent to the Flask API on the web server which can then be relayed to the client over the Python Web Socket the client initiates with the server upon startup. Finally, the HTML site hosted on the web server with Apache displays the incoming sensor data to the user.

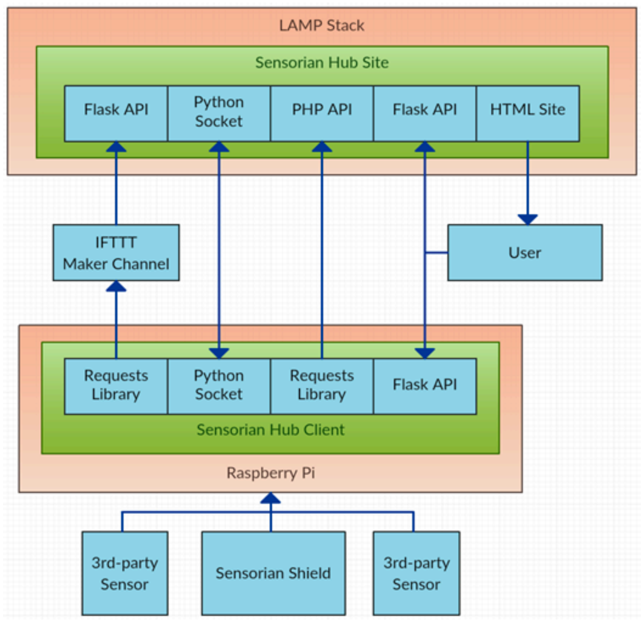


Fig. 6. System architecture diagram.

For the website, a user registers with the site along with the hardware id of their Raspberry Pi. We use the hardware id as a unique identifier for the Sensorian when the JSON data is sent to determine if it belongs to a user in the database. Any requests from non-registered users/devices are simply discarded by the API. When the user logs into the website their data can be shown in either a tabular or graphical format, with the graph sourcing data in real time from new rows in the database.

The site is built on a PHP backend for communicating between the clients and the database. The tables and graphs use AJAX and jQuery scripts to asynchronously update their data from the database. The user can also remotely reconfigure and command the Raspberry Pi through the use of HTTP requests on the local network or over the net through the use of the site as a relay.

The need for a relay server was quickly realized while working behind a router over which we had no control of port forwarding or IP address assignment. In order to overcome this issue, the site can act as a middleman in these transactions by receiving the request itself and forwarding them on to the client. This is achieved through the use of Python on the server as well to listen for TCP socket connections initiated by a client Pi. Once this link is established, it allows for two-way communication between the server and client, and by extension the rest of the web.

V. EVALUATION AND RESULTS

We gauged the success of our solution on a variety of metrics. Our tests for the modularity and customizability of the client tie in closely with the tests of its reliability and robustness. This is due to the fact that through testing the simultaneous operation of the various sensors we encountered numerous bugs preventing concurrent processing. Besides the obvious need for threading locks on the global variables potentially being accessed by individual threads simultaneously, we also uncovered the need for a thread lock on the Sensorian firmware itself. Since the firmware utilizing the I2C and SPI interfaces is not thread-safe, we had to ensure that only one thread was accessing the interfaces at any given time. Another modularity issue discovered was that the light sensor needed to be reinitialized every time it was used since other sensors would seemingly overwrite some of its necessary information when called themselves.

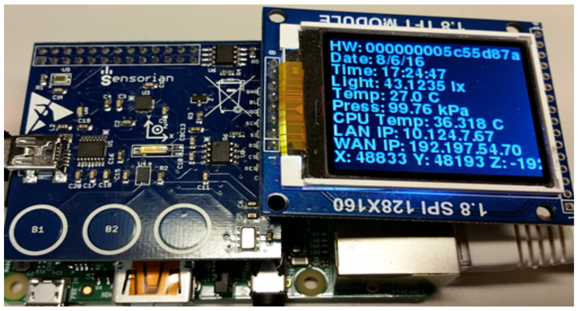

HW: 000000005c55d87a
Date: 8/6/16
Time: 17:24:47
Light: 43.1235 lx
Temp: 27.0 C
Press: 99.76 kPa
CPU Temp: 36.318 C
LAN IP: 10.124.7.67
WAN IP: 192.197.54.70
X: 48833 Y: 48193 Z: -19

Fig. 7. Sensorian LCD displaying sensor data.

As for the robustness of the code, we were able to take hard measurements of the power consumption, CPU/Memory usage, and run extended tests for reliability. Using "top" in the command line we measured the CPU and memory usage of the client to ensure it was running efficiently. As can be seen in Table I, using our default settings the Raspberry Pi B+ averaged 4.2% usage. Additionally, the Pi 2 used around 1.3% CPU, with both allocating around 20 MB of RAM to the process. Most interestingly was the proof that the extra cores of the Pi 2 seemed to play a significant role in reducing overall CPU usage due to the multithreaded nature of our application more than just the simple clock speed boost it received.

Table I. Client CPI usage – Display Off vs. ON

Display State	Virtual Memory	Reserved Memory	Shared Memory	CPU Usage	Memory Usage
Off	166484k	20704k	7944k	4.2 %	4.7 %
On	166484k	20704k	7944k	35.4 %	4.7 %

As for networking usage, a POST request sending data to the database server was determined to be a consistent 486 bytes by using tcpdump and Wireshark [19] for packet analysis. The strict JSON format of the request results in very predictable network usage. Unless the update rate was increased significantly, bandwidth and power consumption as a result of the requests is unlikely to be a concern.

Through the use of a USB digital multimeter as shown in Fig. 8, we were able to get a reading of about 1.613 W using our default settings on the Pi B+. It was only after enabling the display that you could see some spikes to 1.701 W. This measurement directly relates to the CPU and network usage of the Pi, with readings being significantly higher at lower polling rates and before we implemented multithreading. This low power consumption, which is nearly indistinguishable from the idle power consumption of about 1.568 W, would allow the client to run even on battery power for a significant amount of time.

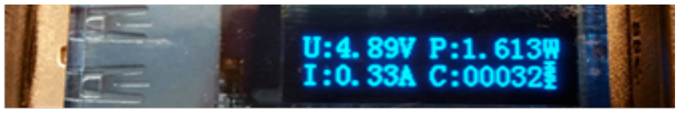
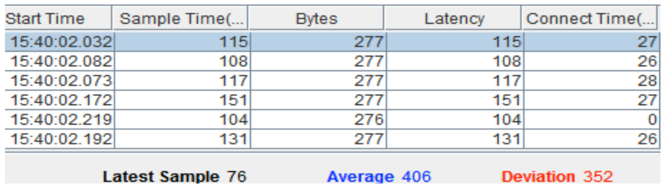


Fig. 8. Client power consumption on Pi B+.

Since each part of the Sensorian Hub uses very fundamental communication protocols, there is minimal load added on top of the base protocol overhead. To demonstrate this, load testing was performed on the Site and Database using Apache JMeter [20] to simulate requests from several clients at once. The Site and Relay were installed on an Ubuntu Virtual Machine on Microsoft Azure [21] for these tests. Using their absolute lowest tier of hardware, classified as Basic A0, the Site was able to handle requests from in excess of 100 clients simultaneously, with latency scaling approximately linearly to the number of clients as can be seen in Fig. 9. This is to be expected based on the load placed upon the testing software, with the Basic A0 server not experiencing anything more than about 10% load on its CPU.

Table II. Apache JMeter results



Start Time	Sample Time(...	Bytes	Latency	Connect Time(...
15:40:02.032	115	277	115	27
15:40:02.082	108	277	108	26
15:40:02.073	117	277	117	28
15:40:02.172	151	277	151	27
15:40:02.219	104	276	104	0
15:40:02.192	131	277	131	26

Latest Sample 76 **Average** 406 **Deviation** 352

VI. APPLICATIONS AND EXAMPLES

With full IFTTT integration in both directions now through the use of their Maker channel and the relay server, a lot of work could potentially be done in creating interesting applications and recipes for use with the Sensorian or other compatible sensors. This would be much easier for a user to implement and have much more potential triggers than other solutions restricted to a single trigger such as email. We have already demonstrated for example that a notification could be sent if the light level crosses a certain threshold for use in automating lights or security. Other possibilities explored in other works include maintaining the temperature of a room or receiving a notification when the laundry is finished when the accelerometer detects no movement. The scalability and relative inexpensiveness of these devices allow us to perform all these tasks and the user to customize them based on need.

Some examples that come with the Sensorian Hub Client and are explained in the included documentation are the light level example and the door sensor example. These are two simple applications utilizing just a single sensor each to achieve their goals. The light level demo maintains a certain ambient light level through sensing the current value and adjusting the brightness of a light to compensate for changes. The door sensor demo uses a magnetometer and a magnet to detect when a door is opened and triggers and alert or action as a result. These examples, while simply programmed for a single location, give great insight into how these could be scaled to larger settings. In the case of the light example, taking the average of multiple light sensors to eliminate anomalies, or for the door sensor example, setting up a multi-door security system for the home.

VII. CONCLUSION AND FUTURE WORK

The Sensorian Hub is designed to be a solution that can be easily adapted to the user's needs, and the less work the user needs to do to achieve this goal the better. The website could stand to benefit from additional features, such as an interface to modify the remote configuration, a unified graphing experience using strictly Flot as opposed to a combination with SmoothieCharts, and the ability to delete data or download it to a CSV file. These sorts of improvements would allow for easier integration with existing tools for big data analysis and machine learning with countless sensors feeding the data.

Finally, while currently based almost entirely on the Sensorian as the solitary source of data for the client, the program has been designed from the ground up to be modular and flexible. Adding support for other sensors to the client is the main work to be done to make our project reach its full potential. Being able to support other shields, hats and individual sensors with the single client and server would be an incredibly useful addition that would truly demonstrate the power and flexibility of the Raspberry Pi as an IoT device.

ACKNOWLEDGMENT

This work is supported by a Canadian NSERC Discovery grant. The authors would like to thank Saurabh Mishra and Abishek Kailash for their help with an earlier and much simpler implementation of the platform as part of the project for the Cloud Computing course (SOFE 4870U) at UOIT.

REFERENCES

[1] "What is a Raspberry Pi?", Raspberry Pi, 2016. [Online]. Available: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/. [Accessed: 08- Jun- 2016].

[2] "Sensorian", Sensorian.io, 2016. [Online]. Available: http://sensorian.io/. [Accessed: 08- Jun- 2016].

[3] Q. Mahmoud and D. Qendri, "The Sensorian IoT platform", 2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 286 - 287, 2016.

[4] S. Banerjee, D. Sethia, T. Mittal, U. Arora and A. Chauhan, "Secure sensor node with Raspberry Pi", IMPACT-2013, pp. 26 - 30, 2013.

[5] S. Ferdoush and X. Li, "Wireless Sensor Network System Design Using Raspberry Pi and Arduino for Environmental Monitoring Applications", Procedia Computer Science, vol. 34, pp. 103-110, 2014.

[6] "RFC 7252 - The Constrained Application Protocol (CoAP)", Tools.ietf.org, 2016. [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 08- Jun- 2016].

[7] V. Vujovic and M. Maksimovic, "Raspberry Pi as a Wireless Sensor node: Performances and constraints", 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1013 - 1018, 2014.

[8] SensorianHubClient, "sensorian/SensorianHubClient", GitHub, 2016. [Online]. Available: https://github.com/sensorian/SensorianHubClient. [Accessed: 30- Jun- 2016].

[9] SensorianHubSite, "sensorian/SensorianHubSite", GitHub, 2016. [Online]. Available: https://github.com/sensorian/SensorianHubSite. [Accessed: 30- Jun- 2016].

[10] S. Jain, A. Vaibhav and L. Goyal, "Raspberry Pi based interactive home automation system through E-mail", 2014 International Conference on Reliability Optimization and Information Technology (ICROIT), pp. 277 - 280, 2014.

[11] V. Vujović and M. Maksimović, "Raspberry Pi as a Sensor Web node for home automation", Computers & Electrical Engineering, vol. 44, pp. 153-171, 2015.

[12] O. Laursen, "Flot: Attractive JavaScript plotting for jQuery", Flotcharts.org, 2016. [Online]. Available: http://www.flotcharts.org/. [Accessed: 08- Jun- 2016].

[13] J. Walnes, "Smoothie Charts: Ten Minute Tutorial", Smoothiecharts.org, 2016. [Online]. Available: http://smoothiecharts.org/tutorial.html. [Accessed: 08- Jun- 2016].

[14] "IFTTT", IFTTT / Connect the apps you love, 2016. [Online]. Available: https://ifttt.com/. [Accessed: 08- Jun- 2016].

[15] "Requests: HTTP for Humans — Requests 2.10.0 documentation", Docs.python-requests.org, 2016. [Online]. Available: http://docs.python-requests.org/en/master/. [Accessed: 08- Jun- 2016].

[16] "Flask (A Python Microframework)", Flask.pocoo.org, 2016. [Online]. Available: http://flask.pocoo.org/. [Accessed: 08- Jun- 2016].

[17] "Flask-RESTful — Flask-RESTful 0.2.1 documentation", Flask-restful-cn.readthedocs.io, 2016. [Online]. Available: http://flask-restful-cn.readthedocs.io/en/0.3.4/. [Accessed: 08- Jun- 2016].

[18] "Welcome to Flask-HTTPAuth's documentation! — Flask-HTTPAuth documentation", Flask-httpauth.readthedocs.io, 2016. [Online]. Available: https://flask-httpauth.readthedocs.io/en/latest/. [Accessed: 08- Jun- 2016].

[19] "Wireshark · Go Deep.", Wireshark.org, 2016. [Online]. Available: https://www.wireshark.org/. [Accessed: 11- Jul- 2016].

[20] "Apache JMeter - Apache JMeter™", Jmeter.apache.org, 2016. [Online]. Available: https://jmeter.apache.org/. [Accessed: 11- Jul- 2016].

[21] "Microsoft Azure: Cloud Computing Platform & Services", Azure.microsoft.com, 2016. [Online]. Available: https://azure.microsoft.com/. [Accessed: 11- Jul- 2016].