ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE
Paper ID #37328

Enabling Generic Sensing Devices to use LoRa Communication

Ciprian Popoviciu (Assistant Professor)

Dr. Ciprian Popoviciu, East Carolina University – Assistant Professor Dr. Ciprian Popoviciu has over 20 years of experience working in technical and leadership roles in the IT industry. He is an industry-recognized domain expert in IPv6 who has worked with large service providers, enterprises, and governments. Popoviciu founded Nephos6, which did groundbreaking work OpenStack for IPv6, authored two books on IPv6, and has worked on IPv6-related internet standards and patents. He is an IPv6 Forum Fellow, IPv6 Hall of Fame 2019 inductee, and a technology expert for the European Commission. His research focuses on next generation infrastructures and IOT. Dr. Popoviciu completed his Executive MBA at Kenan-Flagler Business School, University of North Carolina at Chapel Hill. Dr. Popoviciu is currently an assistant professor at East Carolina University's College of Engineering and Technology, where he teaches and participates in cyberinfrastructure research.

Colby Lee Sawyer

Colby Sawyer, East Carolina University – Software Developer

Enabling Generic Sensing Devices to use LoRa Communication

Colby Sawyer, Ciprian Popoviciu

Abstract: There is a rapidly increasing need for real-time sensing data; data typically collected by devices deployed in remote locations, inaccessible by standard wireless and cellular technologies at scalable costs. With the emergence of LoRa wireless technology as a viable data communication methodology, there is a clear opportunity to instrument the environment at scale with sensors to collect real-time data at low cost and low power budgets. In this paper, we cover the approach used to develop an edge device capable of interfacing with any environmental sensor to transmit collected data via LoRa to a cloud-based data collection service. We document the development of the physical edge node, the development of a methodology to generalize connectivity to sensing devices, the development of software capable of interfacing with the connected sensors, the development of software capable of translating data into meaningful information, and the development of software to send/receive data over LoRa to a cloud-based infrastructure monitoring service. We aim to streamline the instrumentation of this device to enable any scientific group to easily set up an instrumentation infrastructure to collect data over large geographical footprints.

Key words: IOT, LoRa, Sensors, Monitoring.

1. Introduction

The emergence of the Internet of Things (IoT), a collection of technologies, products, and services facilitating easy, inexpensive deployment and management of devices to instrument environments and processes. By combining the data collection and data-based actuation provided by IoT infrastructures with analytics, machine learning and artificial intelligence, the generated value is clear with respect to data gathering, gaining insights, and driving workflow optimizations.

New wireless communications technologies optimized for range, power consumption, interference avoidance and finally for cost are making it easier to build IoT infrastructures. Deploying and managing sensors on these infrastructures is dependent on the communications interfaces built in the sensors. Most traditional sensors include a serial interface SDI-12 (Serial Digital Interface at 1200 baud) [1] or I2C [2][3] and at times a short-range communication interface such as Bluetooth for proximity reading of logged data. While newer sensors are catching up with long-range communications technologies integrating interfaces such as LoRa, WiFi, and cellular, the diversity in technology options makes modular designs, where the sensing and communications pieces are distinct, more practical.

Additionally, the practitioners who are most interested in leveraging sensing infrastructures and innovating on top of them are typically specialized on the technology and science residing at the two ends of the IoT environments: Sensing and Analytics. Researchers, Educators, and Practitioners are often faced with the challenge of connecting existing or new types of sensors, work that takes away from the focus of their research: development of new sensing methods or analyzing sensor data. Enabling users to simply deploy their sensors and gain access to the collected data would optimize the use of their own time and funding.

In this paper we detail the work of designing and implementing an interface, called Communications Shim (CS), facilitating the integration of commercial and experimental sensors with a serial interface into a LoRaWAN (Long Range Wide Area Network) infrastructure. This project is part of a larger initiative within the Technology Systems department at East Carolina University [4] to develop the Campus-as-a-Lab (CaaL) platform to enable instrumentation in support of researchers, educators, students, and facilities managers. The CaaL architecture is described in a separate paper [5]. This paper covers the approach, design, and implementation of the Communications Shim and its testing within the CaaL infrastructure.

2. Problem Definition and Requirements

IOT infrastructures consist of three main domains (Figure 1). Sensing covers all aspects related to the ways in which data such as water level, temperature, humidity is collected. The sensing domain is also focused on local data management. Connectivity covers the various technologies and methods to connect sensors with data collectors and connect sensors amongst themselves. The connectivity domain also addresses the manageability of the sensors. Finally, Data and Analytics covers all aspects of storing and managing collected data, covers the processes to analyze data, deliver insights and enabling action based on the collected data. This domain is responsible for data lifecycle management and access to data analytics tools.

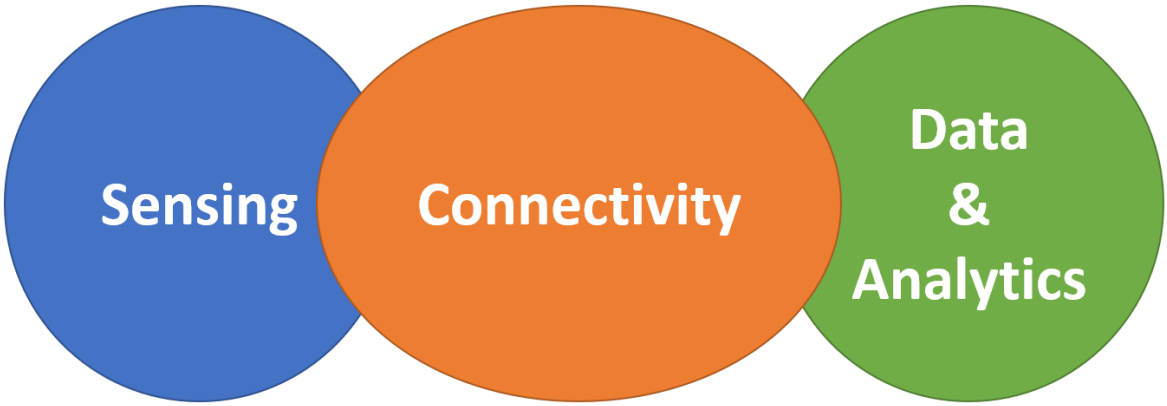


Figure 1. Main domains of a sensing platform [3].

Most researchers focus on the sensing, data & analytics or both domains. In the sensing area, researchers are investigating and developing new sensing technologies or way to lower the cost of existing sensors leading to new sensors. At the same time, in their goal to deploy instrumentation, researchers would like to deploy traditional sensors in an IoT environment and eliminate the need to periodically visit sensors in the field to download collected data. Since each sensing project might be best served by a different communication technology, it is important to facilitate easy integration of sensors into available communications technologies.

In the context of the CaaL project we interviewed the key users and stakeholders to gather requirements for the solution. One of the key requirements was to facilitate the integration described above, specifically, this integration should:

- **Accommodate Various Types of Long-Range Communications Interfaces –** Integrate sensors in various communications infrastructures such as: LoRa [6], Cellular [7], CBRS (Citizens Broadband Radio Service) [8], etc.
- **Accommodate Multiple Sensing Device Types –** Interface commercial and development sensors using standard serial interfaces.
- **Provide Edge Computing and Data Store Resources –** Provide compute and memory resources close to the sensing devices to facilitate data management and edge computing.

Within the CaaL architecture (Figure 2) we identified a sensor integration module we called: Communications Shim (CS).

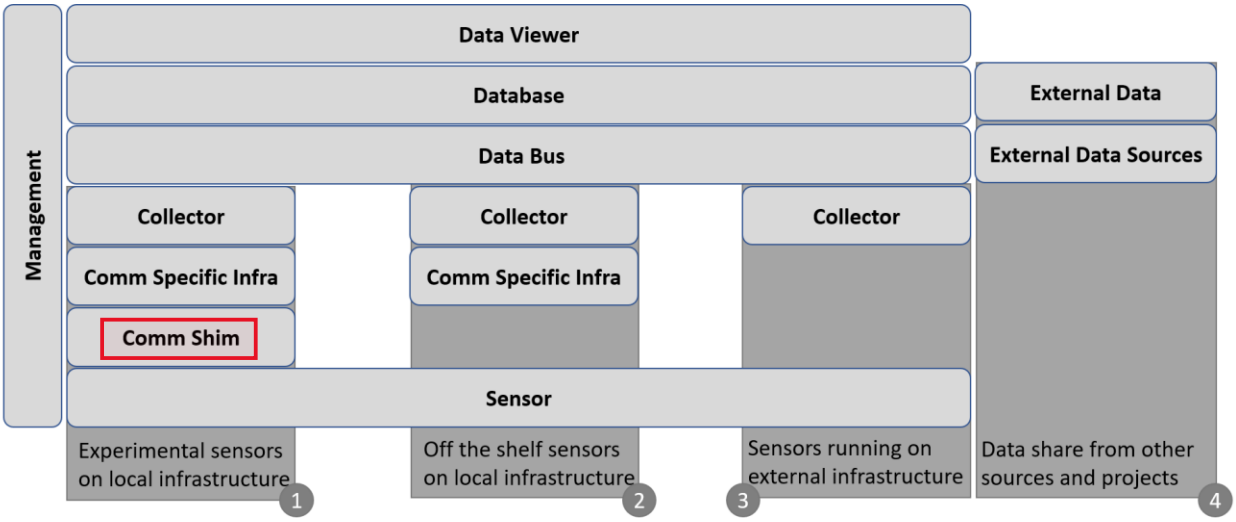


Figure 2. High-Level CaaL Architecture and Use Cases [3]

In the following section we described the implementation of CS at hardware and software level.

3. Implementation and Results

Creating a device that can communicate via LoRa technology and support a very wide range of commercial and experimental sensing devices presented several challenges related to hardware and software options. The hardware selection was driven by performance, power consumptions restraints, portability, and overall product ease-of-use. Several software components are required to facilitate the interface between interfaces while running on embedded platforms with limited resources. In this section we describe the hardware setup, software elements and security provisions that were integrated and developed to create the production level Communication Shim (CS).

3.1. Hardware Selection

Originally, the CS was intended to operate on a low-power platform, to align with one of the key benefits of LoRa. During the process of developing the supporting code, the sensor connectivity, and translating the sensor readings to human-readable format; a necessary pivot was identified, a pivot that turned the project away from the ultra-low-power platforms which did not have enough memory to support all intended CS functionality. The first iteration focused on the Adafruit Industries “Feather” line of platforms with a highly supported set of integrated LoRa radios [9]. The Feather M0 is a flexible platform, and it facilitated the process of learning how to connect devices to a LoRa gateway. The issue that eventually made the Feather platform unsuitable was the limited amount of usable memory.

Next, we focused on the Mayfly platform [10] and mDOT LoRa module [11] integration which proved difficult due to a lack of community support and issues in communication protocol configurations.

The Raspberry Pi 0 platform, developed by the Raspberry Pi Foundation [12] combined with LoRa radio bonnets, provides integration with various serial communication protocols, and has a sufficient memory available for the needs of CS. The Raspberry Pi 0 doesn’t meet the original expectations of low-power consumption, but the benefits of the larger compute resource outweighed the concern of increased power consumption. Further encouraged by the evidence from D. Patnaik Patnaikuni and M. Maksimovi who found “based on the specs and performance analysis Raspberry Pi definitely emerges as a winner when it comes to satisfying most of functional requirements of an IoT systems’ basic blocks” [13][14] Also, the very nature of the rapid prototyping and community support for the Raspberry Pi foundation is highly

advantageous. While PI 0 supports the simplest version of CS, it can easily be scaled up to more powerful PI boards. The CS that was created with the Raspberry Pi 0 and Adafruit LoRa Radio Bonnet [15] meets the expectations set for CS.

Figure 3.0: Raspberry Pi 0 and Adafruit LoRa Radio Bonnets (separated for example)

Raspberry Pi 0 has an extensive level of support for interfacing with serial devices on the native GPIO, enabling the initial use of I2C sensors. The CS can communicate with simple I2C environment sensors (Temperature, Humidity, Atmospheric Pressure) and translate the serial protocol to LoRa transmittable packets. L. Barik noted during their experiment with a very similar system, that “The system also provides a corrective movement or decision-making system … also allows the consumers to research the correct modifications within the surroundings and for taking possible action” [16]. At this stage of the CS development, this is exactly what we wanted to provide to our process. Testing proved this operation was stable and replicable, leading to extending the experiment to different sensors that utilize communications protocols other than I2C. We identified, based on a user sample, that many environmental sensors communicate via the SDI-12 protocol [1].

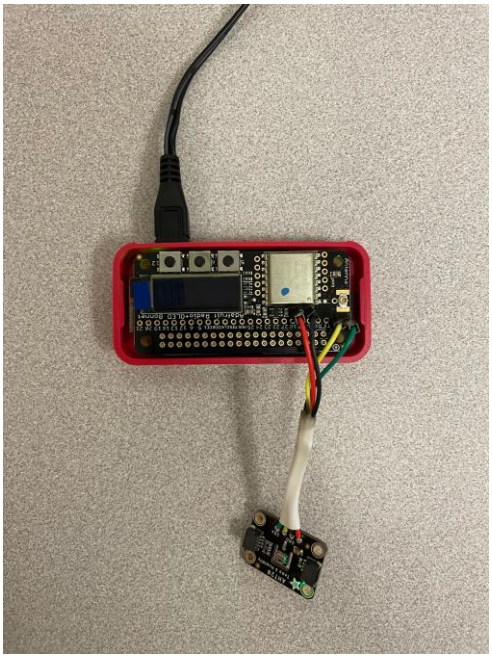


Figure 3.1: CS connected with I2C Sensor

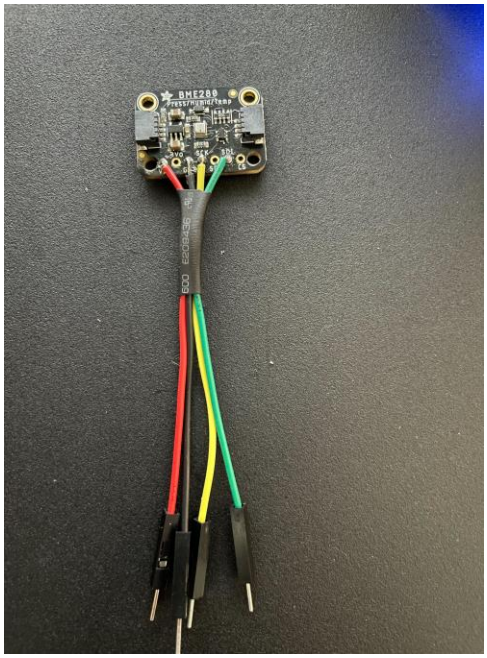
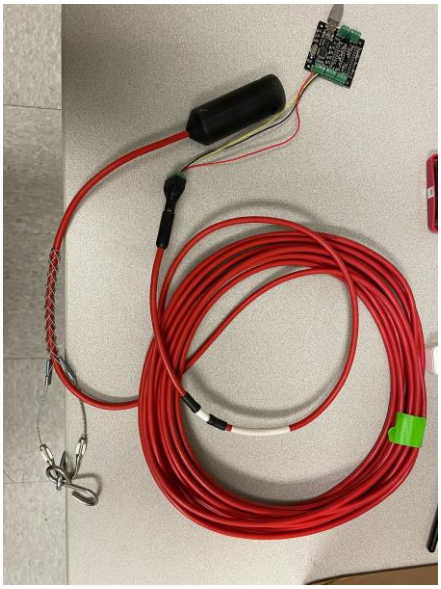


Figure 3.2: BME280 I2C Environmental Sensor

The limitation experience in this integration is that the LoRa Bonnet [15] required the use of GPIO pins typically used by other serial interfaces such as SDI-12. This issue was solved by introducing an external SDI-12 to USB translation board from Dr. John Liu [17]. This external board allowed the CS to interface with up to four SDI-12 communicating devices while also allowing for external power to be passed directly to the sensors. The CS was now able to support the two most common environmental sensor communication protocols (I2C, SDI-12).



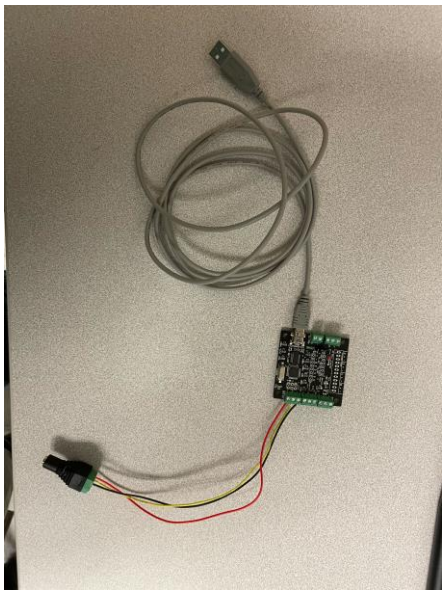


Figure 3.3: Hydros Water sensor connected to SDI-12 External board with 3-Channel audio input to USB

3.2. Software

With the hardware configuration defined, the work started on software implementation supporting protocol translation between the connected interfaces. The decision to use the Raspbian OS [18] was based primarily on its ability to rapidly flash the device while also gaining access to all the standard tools that one could expect in a UNIX-like environment. A useful set of tools were developed by the Raspberry Pi Foundation to use and secure the various serial interfaces natively supported on the board. These tools became instrumental in laying the foundation for the CS software suite.

Along with the Raspbian OS, additional code had to be developed to communicate with the Adafruit LoRa radio and the various sensors. There is a community project sponsored by the LoRa Alliance [19] to support wide ranges of LoRa devices in various languages such as Python, C++, and Java. This community produced a set of Python classes that would become instrumental in our development of a stable library capable of communicating with the Adafruit Radio Bonnet [15] based on the RFM95W module (Figure 3.4). Changes made to the community code base centered around protocol characteristics, adaptation to the US LoRa frequency band and multiplexing capabilities [20,21].

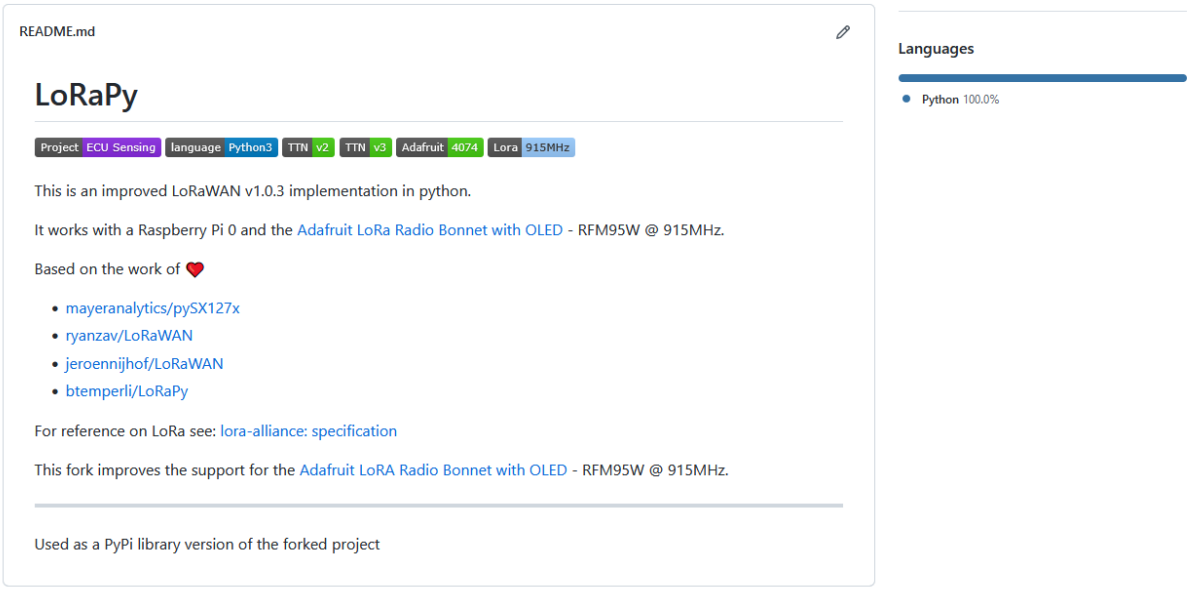


Figure 3.4: GitHub Repository of LoRaPy library source

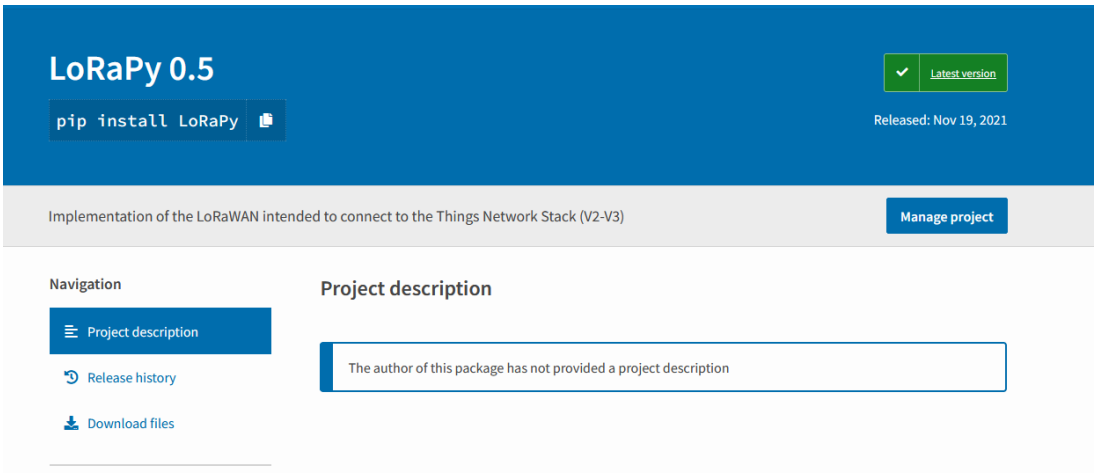


Figure 3.5: PyPi location of LoRaPy library

With the LoRa communication software developed and tested, the next step was to develop the code needed to communicate with the serial devices, report back their readings, translate the readings into a human-readable format and package them to be used in the LoRa transmissions. The initial efforts to include sensors based on the I2C protocol leveraged product libraries [22]. To integrate the SDI-12 communication protocol we leveraged a portion of the Python code provided by Dr. John Liu [17][23]. The effort to support a generic communication protocol with the external board essentially leveraged a set of pre-existing open-source Python libraries, community-developed to communicate with serial devices.

data.py

```
# SDI12 - USB Adapter by Liudr data collection script. Intended to be used with Raspberry Pi LoRa Driving code (https://github.com/ECU·
# Author: Colby Sawyer 1-5-2022

#TODO Add support for more devices
from devices.hydros import Hydros

#TODO Add data connection testing (multiple device support)
#TODO Add device specific classes (allow for specific data representation)

#//=======================================
def get_data():
    """Main driver to fetch most recent data and return in the form of a bytearray for transmitting over LoRA

    Returns:
        bytearray: bytearray containing encoded data from the logger (intended for LoRa usage)
    """
    sensor_data = Hydros().get_data()

    return sensor_data
#//=======================================

get_data()
```

Figure 3.6: SDI External Board driving code example

Finally, to address embedded platform constraints, purpose-built software was developed to manage data regardless of interfaces (SDI12, I2C, etc). To test the capabilities of the CS, a Hydros 21 [24] water level sensor was used to model a new Python class that could communicate and translate its readings (see Figure 3.2). With the development of this new Hydros specific library the CS was able to communicate with the external board, send serial commands to the Hydros sensor and retrieve the response.

hydros.py 1 ×

devices > hydros.py > ...

```python
"""
`hydros`
================================================================================
Python library for Hydros21 or Decagon CDT-10 water level sensor.
* Author(s): Colby Sawyer
Implementation Notes
--------------------
**Hardware:**
* `Hydros 21 Conductivity, Temperature, and Depth Sensor <https://www.metergroup.com/en/meter-environment/products/hydros-21-water-level-sensor-conductivity-t

**Software and Dependencies:**

"""
import serial.tools.list_ports
import serial
import time
import re

class Hydros:
    """Driver for Hydros 12 or Decagon CDT-10 water level sensors
    """
    water_depth = 0
    temperature = 0
    electrical_conductivity = 0

    def __init__(self, water_depth=0, temperature=0, conductivity=0):
        """Initialize instance of sensor

        Args:
            water_depth (int, optional): [description]. Defaults to 0.
            temperature (int, optional): [description]. Defaults to 0.
            conductivity (int, optional): [description]. Defaults to 0.
        """
        self.water_depth = water_depth
        self.temperature = temperature
        self.electrical_conductivity = conductivity

    def parse_reading(val):
        """Takes string of values from sensor and parses them into String list
```

Figure3.7: Hydros specific code example

The Hydros sensor integration proved that CS meets the original functional requirements. To achieve generic adherence to standards, additional community driven packages were included in the code repository. We applied a modular approach to developing the CS code to facilitate, extensibility, continuous-development, and integration.

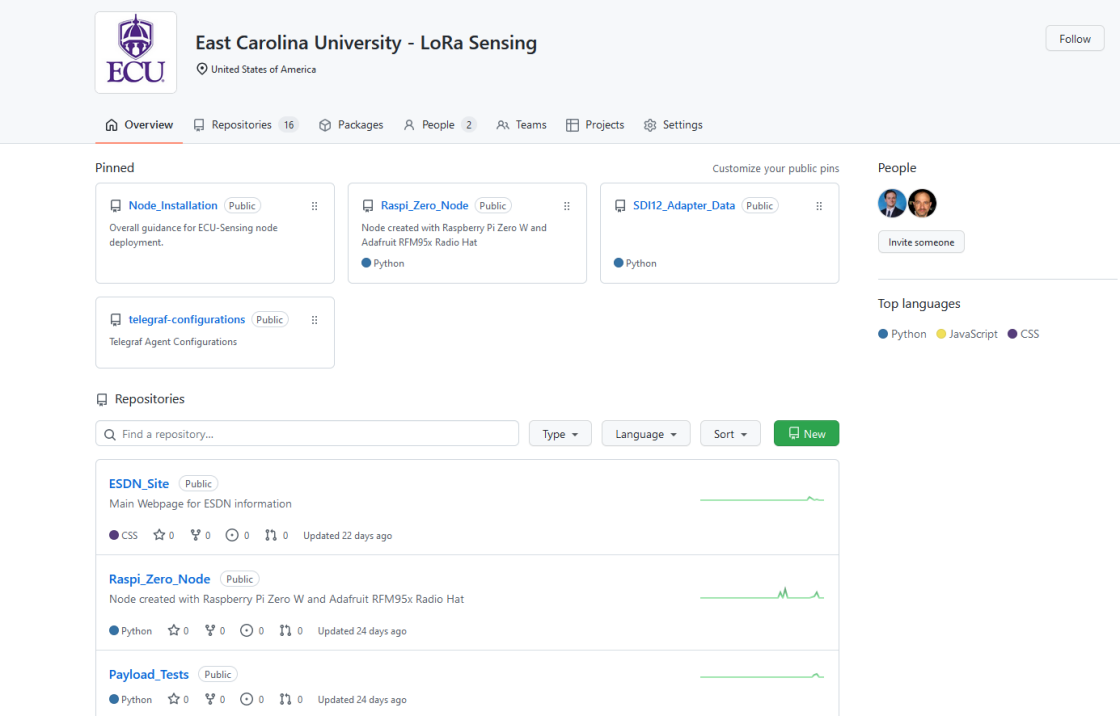


Figure 3.8: GitHub project management dashboard

To enable the continuous collection of data at configurable time intervals, a cron scheduler [25] is used. The cron scheduler is a command-line utility for Unix-like operating systems that will issue a job to run continuously at regular intervals set by the user. In the case of the lab environment, the cron scheduler was utilized to run the CS code every ten minutes. The goal is to have the application manage poling dynamically to facilitate adaptable polling rates.

```
  GNU nano 4.8                          /tmp/crontab.VgHdoR/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
*/10 * * * *  sudo python3 /home/pi/Raspi_Zero_Node/driver.py


                                [ Wrote 24 lines ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo  M-A Mark Text
^X Exit      ^R Read File  ^\ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo  M-6 Copy Text
```

Figure 3.9 Crontab example configuration

To manage the device specific software and facilitate continuous development and community participation, a management framework was implemented.

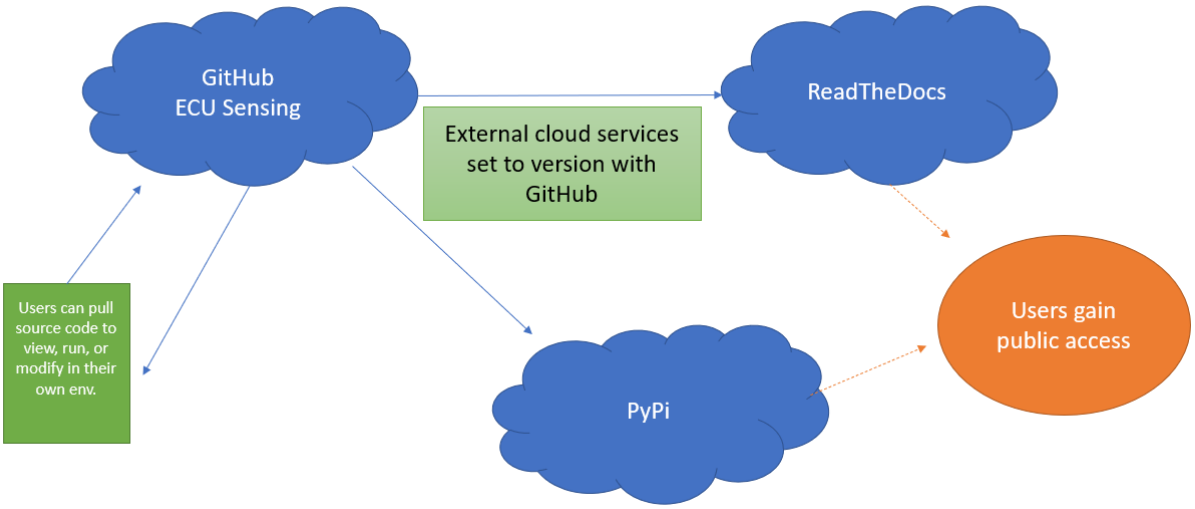


Figure 3.10: Code Management Schema

GitHub is utilized to manage the version control of the entire library allowing the community to contribute additional code in a manageable, controllable way. The Python Package Index (PyPi) [26] allows the developed Python software to be published in a secure, public space as a usable library for any Python application. The packages are compiled on the developed environment using default Python tooling before being published to PyPi using a tool named twine [27]. ReadTheDocs [28] and Sphinx [29] are tools used to manage the documentation process of the codebase. Contributions internal or external must provide written documentation for every contribution to the library and, using Sphinx, those comments can be directly published on a ReadTheDocs hosted website. The importance of early forced adoption of documentation styling was proven by C.J.Satish and M.Anand as they noted that “Enforcing higher documentation standards and stringent reviews on documentation content and structure can solve much of the problems with documentation even without the tool support” [30]. Since they also concluded that it is important for specific working projects to tailor their environment to not hinder development, we focused on incorporating both styling procedure and tooling meant to advance development efforts without forcing developers to get trapped spending more time on documentation than development. Our tooling in this case, (Sphinx) adheres to a very commonly practiced documentation style, like that of Google’s Python docstring formatting guidelines meant to be simple and quickly implementable [31]. An example of our defined documentation style is shown in Figure 3.8.

```
"""[Summary]

:param [ParamName]: [ParamDescription], defaults to [DefaultParamVal]
:type [ParamName]: [ParamType](, optional)
...
:raises [ErrorType]: [ErrorDescription]
...
:return: [ReturnDescription]
:rtype: [ReturnType]
"""
```

Figure 3.11: Docstring formatting example

3.3. Security Considerations

In the context of the CaaL architecture, the end-to-end solution implements the best practices for securing cyberinfrastructures. Transport is secured using protocol specific security features and by applying layered security to harden the infrastructure. To avoid making the CS the weak link in the solution security architecture, the CS had to be explicitly hardened against security threats.

To secure the LoRa interface we implemented the LoRaWAN security options. According to the LoRaWAN specifications there are two activation modes recommended for all LoRa communication devices. ABP (Activation by Personalization) and OTAA (Over the air Activation) are responsible for managing the verification of each endpoint during the LoRa transmission [32]. In this case, the gateway is responsible for communicating with the stack to properly verify each device that communicates with it by using fixed or dynamic keys. It is generally recommended to use the OTAA activation mode as it relies on dynamic keys rather than the less secure fixed keys utilized by the ABP mode. The incorporation of software that supports both ABP and OTAA happened upon to original creation of the CS software-suite. This means that our CS utilizes the more secure OTAA mode when communicating with the gateway, but ABP could be used for test or lab environments. However, its detailed by Aras et al “most LoRaWAN security measures such as the key management and frame counters need to be implemented and taken care of by developers or manufacturers. Therefore, poor implementation also may put end-devices and gateways in danger” [33]. We knew that the CS device would need to properly deploy the LoRaWAN security protocol to integrate with the CaaL and this why we took care in implementing OTAA for all deployable CS devices.

Securing the Raspberry Pi was based on the principles of host hardening. The Linux-based OS is secured by properly separating user permissions and by creating secure authentication for the admin user. The CS adheres to this standard by utilizing a single admin user that is configured with a 12-character cryptographic password. The Raspberry Pi also needed to have its many unused interfaces disabled. Utilizing a Raspbian configuration tool we were able to manage any interface supported on the device [34].

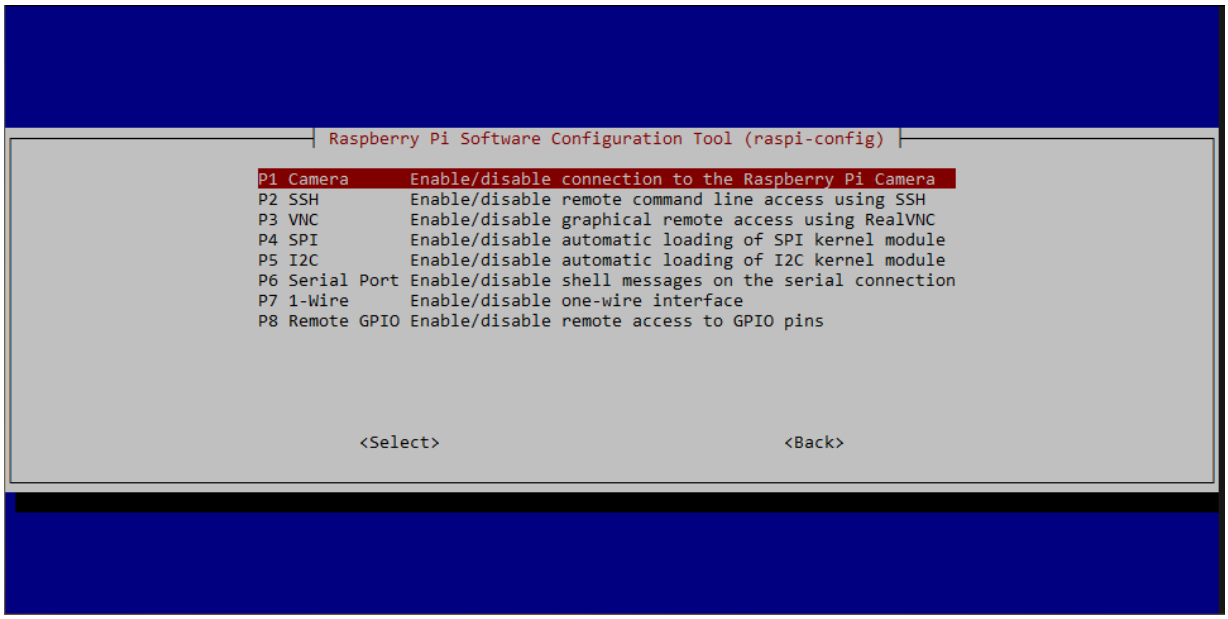


Figure 3.12: Raspbian Configuration Tool

At the time of this writing, the CS does not provide Layer 3 connectivity, so security measures are limited to OS, Layer 1 and Layer 2.

3.4. Case Development

To properly secure and protect the CS hardware and the access to its interfaces while making it easy to connect sensors and power sources, a separate, dedicated project was launched. This interdisciplinary project involves the CS development team, researchers working on a wide range of sensors and a team of students majoring in Industrial Engineering and Design.

The requirements identified by the project team are listed below:

- Accommodate the CS hardware (Raspberry PI, Aggregating Board, Antennas)
- Accommodate the form factor of the various Raspberry PI boards
- Provide easy, secure, and waterproof access to the CS interfaces
- Allocates proper space for a battery or other forms of external power
- Cost efficient

The design work will combine rapid prototyping with continuous integration to achieve a product that both meets the above requirements and the 3D printing production considerations. The design work will be done in Autodesk Inventor 2019 with CAD files of the mounting and other hardware imported via McMaster-Carr. The printing of the prototype will use a Stratasys Dimension SST 1200ES [35] with GrabCad Print [36] dedicated slicing software to estimate print times and material. We previously found this to be an efficient methodology for case development in a separate study [37]. The results of this aspect of the project will be reported in a future publication.

4. Conclusions and Future Work

The development of the Communication Shim was completed to support the interface of SDI-12 and I2C with LoRaWAN. The CS was tested extensively, and it was integrated in our Campus as a Lab deployment. The CS was used in proof-of-concept environments and shown to operate well. We are now proceeding to deploy it broadly in support of education, research, and campus operations.

The CS is more than an enabler of sensor integration, it is also a development platform that will be used for education purposes, particularly in scripting. Students will have the opportunity to develop integration scripts for sensors that need to be onboarded for various projects.

The CS will continue to evolve according to a clear roadmap. To further increase the usability of the CS, we plan to add local caching, additional communication interfaces, monitoring capabilities, security with a PUF (Physical Unclonable Function) [38], and a field provisioning/troubleshooting tool capable of communication via Bluetooth. The development of local caching will enable the mitigation of data loss by utilizing the CS software to store small amounts of data. This will allow researchers multiple data extraction options outside of the standard wireless communication protocol. This becomes critical when errors occur, and complete datasets must remain intact.

Additional communication protocol support will become necessary as we gain more insight from research groups on their intended sensors. We plan to support any sensor that can interface with the CS but there will be some slight configurations needed to implement communication protocols outside of SDI-12 and I2C.

We plan to tool the CS to support multiple monitoring tools to ease the process of troubleshooting and maintenance. This will become instrumental during the initial deployment phase and will allow us to maintain the deployed CS devices more efficiently as we expand the infrastructure. This will also provide the administration group invaluable information for various device usage metrics, error reporting and lifecycle management.

To further secure the CS devices we will put a plan in place to introduce PUF (Physical Unclonable Functions) technology “closing any security loopholes and providing a defense against attacks” [38]. As

previously discussed, we utilize the OTAA (Over The Air Activation) activation mode of LoRaWAN to protect against some security vulnerabilities, but OTAA is not a perfect protocol. The use of the PUF technology is important to protect against the proven OTAA vulnerability to sniffing attacks detailed by a research group from Ajou University. They stated “We can check all of the contents in join request message including frequency and SF (Spread Factor) information without decryption process. In the same way as this practical experiment, attackers could look inside the packet without any hindrance and cause serious damage by abuse it” [39]. PUF usage will enable the CS to be further verified before it’s allowed to communicate, over any wireless protocol, to the datastore. This will become instrumental as we focus on security with the incorporation of multiple data sources.

Opening Bluetooth communications on the CS devices will allow us to provide users with tooling capable of troubleshooting the CS and attached sensors while in the field. This will be extremely beneficial as it allows researchers to quickly correct any errors without removing devices and costing hours of lost work and effort. Mitigation of downtime also becomes important as we look to prove the stability of the CS. This adheres to our goal of enabling users of any discipline to deploy their sensors quickly and easily.

References

[1]. Sdi-12.org, *SDI-12 Support Group*. [Online] Available: https://sdi-12.org/ [Accessed 26 Jan 2022].

[2]. I2C Info – I2C bus, interface and Protocol," *I2C Info – I2C Bus, Interface and Protocol*. [Online]. Available: https://i2c.info/. [Accessed: 28 Jan 2022].

[3]. Mankar, J., Darode, C., Trivedi, K., Kanoje, M., & Shahare, P. "REVIEW OF I2C PROTOCOL" *International Journal of Research in Advent Technology*, vol.2, no.1, pp. 474–479. Available CiteSeer http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.429.1402&rep=rep1&type=pdf

[4]. East Carolina University, College of Engineering and Technology, Department of Technology Systems, Information and Computer Technology, Available: https://cet.ecu.edu/techsystems/undergraduate-programs/information-computer-technology/ [Accessed: 28 Jan 2022]

[5]. Popoviciu, C., Sawyer, C., "Building the Campus-as-a-Lab Platform", *2022 ASEE Annual Conference*

[6]. LoRa Alliance LoRa — "LoRa documentation", [Online] Available: https://lora.readthedocs.io/en/latest/ [Accessed: 28 Jan 2022].

[7]. Gaddam, S. C., and Rai, M. K., "A Comparative Study on Various LPWAN and Cellular Communication Technologies for IoT Based Smart Applications," *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)*, 2018, pp. 1-8, doi: 10.1109/ICETIETR.2018.8529060.

[8]. FCC. Mar 10, 2021. "3.5 GHz Band Overview", [Online] Available: https://www.fcc.gov/35-ghz-band-overview [Accessed: 28 Jan 2022].

[9]. Industries, A, 2022, "Adafruit Feather M0 with RFM95 LoRa Radio - 900MHz", [Online] Available: Adafruit Industries. https://www.adafruit.com/product/3178 [Accessed: 28 Jan 2022].

[10]. "Getting Started With the Mayfly Data Logger". [Online]. Available: https://www.envirodiy.org/mayfly/ [Accessed: 28 Jan 2022].

[11]. "Long Range RF Module for Sale | MultiTech mDot". [Online]. Available: https://www.multitech.com/brands/multiconnect-mdot [Accessed: 26 Jan 2022].

[12]. Raspberry Pi Foundation, "Teach, Learn, and Make with". [Online]. Available: https://www.raspberrypi.org/ [Accessed: 25 Jan 2022].

[13]. Maksimović, M., Vujović, V., Davidović, N., Milošević, V., & Perišić, B. 2014. "Raspberry Pi as Internet of things hardware: performances and constraints". *design issues*, vol.3, no.8, pp.1-6.

[14]. Patnaik Patnaikuni, D. 2017. "A Comparative Study of Arduino, Raspberry Pi and ESP8266 as IoT Development Board". *International Journal of Advanced Research in Computer Science*, vol.8, no.5, pp.2350–2352.

[15]. Industries, A. "Adafruit LoRa Radio Bonnet with OLED - RFM95W @ 915MHz". Adafruit Industries LoRa Radio Bonnet. [Online]. Available: https://www.adafruit.com/product/4074 [Accessed: 28 Jan 2022].

[16]. Barik, L. 2019. "IoT based Temperature and Humidity Controlling using Arduino and Raspberry Pi". *International Journal of Advanced Computer Science and Applications*, vol.10, no.9. Available: https://doi.org/10.14569/ijacsa.2019.0100966

[17]. Liu, J., "SDI-12 Sensors USB Adapter User Manual". [Online] Available: https://liudr.wordpress.com/ [Accessed: 28 Jan 2022].

[18]. Raspberry Pi Foundation. “Raspbian OS”. [Online]. Available: https://www.raspbian.org/ [Accessed: 28 Jan 2022].

[19]. LoRa Alliance. “Homepage”. [Online]. Available: https://lora-alliance.org/ [Accessed: 25 Jan 2022].

[20]. Sawyer, C. “GitHub - ColbySawyer7/LoRaPy: LoRaWAN implementation in python”. GitHub. [Online] Available: https://github.com/ColbySawyer7/LoRaPy [Accessed: 28 Jan 2022].

[21]. Temperli, B. “GitHub - btemperli/LoRaPy: LoRaWAN implementation in python”. [Online] Available: https://github.com/btemperli/LoRaPy [Accessed: 28 Jan 2022].

[22]. Adafruit Industries. “GitHub - adafruit/Adafruit_CircuitPython_BME280: CircuitPython driver for the BME280”. [Online] Available: https://github.com/adafruit/Adafruit_CircuitPython_BME280 [Accessed: 28 Jan 2022].

[23]. Liu, J. “SDI-12 USB adapter - Liudr’s Blog”. [Online] Available: https://liudr.wordpress.com/gadget/sdi-12-usb-adapter/ [Accessed: 28 Jan 2022].

[24]. Meter. “HYDROS 21 Water Level Sensor. Meter - HYDROS 21”. [Online] Available: https://www.metergroup.com/en/meter-environment/products/hydros-21-water-level-sensor-conductivity-temperature-depth [Accessed: 28 Jan 2022].

[25]. “crontab(5) - Linux manual page”. [Online] Available: https://man7.org/linux/man-pages/man5/crontab.5.html [Accessed: 26 Jan 2022].

[26]. “PyPI · The Python Package Index”. [Online] Available: https://pypi.org/ [Accessed: 28 Jan 2022].

[27]. “twine 3.7.1 documentation”. [Online]. Available: https://twine.readthedocs.io/en/stable/ [Accessed: 26 Jan 2022].

[28]. ReadTheDocs. “Read the Docs: Documentation Simplified — Read the Docs user documentation 7.1.0 documentation”. [Online] Available: https://docs.readthedocs.io/en/stable/ [Accessed: 26 Jan 2022].

[29]. Sphinx. “Overview — Sphinx documentation”. [Online] Available: https://www.sphinx-doc.org/en/master/ [Accessed: 26 Jan 2022].

[30]. Satish, C. J., & Anand, M. “Software Documentation Management Issues and Practices: A Survey”. *Indian Journal of Science and Technology*, vol.9, no.20, Available: https://doi.org/10.17485/ijst/2016/v9i20/86869 [Accessed: 26 Jan 2022].

[31]. Google. “Google Python Docstring Style Guide”. [Online] Available: https://google.github.io/styleguide/pyguide.html#s3-python-style-rules [Accessed: 26 Jan 2022].

[32]. The Things Stack for LoRaWAN. “ABP vs OTAA”. [Online] Available: https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/ [Accessed: 26 Jan 2022].

[33]. Aras, E., Ramachandran,G.S., Lawrence, P., and Hughes, D., "Exploring the Security Vulnerabilities of LoRa,". *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*, 2017, pp. 1-6, doi: 10.1109/CYBConf.2017.7985777

[34]. The Raspberry Pi Foundation. “Raspberry Pi Documentation – Configuration”. [Online] Available: https://www.raspberrypi.com/documentation/computers/configuration.html [Accessed: 26 Jan 2022].

[35]. “Stratasys SST 1200EX user guide”, [Online] Available: http://www.stratasys.com/-/media/files/documentation/fdm/Dimension-1200es-and-1200/User-Guide/1200es-User-Guide [Accessed: 1 Feb 2022].

[36]. “GrabCa Print”. [Online] Available: https://grabcad.com/print [Accessed: 26 Jan 2022].

[37]. Popoviciu, C., & Lunsford, P. J., & Pickard, J., & Sawyer, C. L., & Drummond, D., & Zynda, Z. R., & Lee, S., & Wear, S. (2020, June), “A Multidisciplinary Project: Deploying Edge Computing to Augment Endpoint Functionality Paper”. *2020 ASEE Virtual Annual Conference*

[38]. Bohara, R., Ross, M., & Popoviciu, C. (in press). “Physical Unclonable Functions for Identification of Large Scale distributed IoT Assets”. *Unpublished.*

[39]. SeungJae N., DongYeop H., WoonSeob H. and Ki-Hyung, K., "Scenario and countermeasure for replay attack using join request messages in LoRaWAN," *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 718-720, doi: 10.1109/ICOIN.2017.7899580.