CI Rainbow: A Flexible WSN for Environmental Data

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Kevin Scrivnor

December 2016

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

[illegible] 01/17/2017

Advisor: Dr. Andrzej Bieszczad Date

[illegible] 01/17/2017

Dr. Jason Isaacs Date

[illegible] Jan. 17, 2017

Dr. David Claveau Date

APPROVED FOR THE UNIVERSITY

[illegible] 1-17-17

Dr. Gary Berg Date

CI Rainbow: A Flexible WSN for Environmental Data

by

Kevin Scrivnor

Computer Science Program

California State University Channel Islands

Abstract

CI Rainbow is a framework for establishing a network of sensors supporting real-time and long-term environmental studies at remote locations such as on Santa Rosa Island in the Channel Island of California archipelago. A variety of sensory nodes have been implemented with the goal of collecting information about the island. A supporting infrastructure that includes communication links and protocols, database storage, OAM&P (operation, administration, maintenance, and provisioning), and access tools has been designed and deployed on an exploratory solar-powered self-sustainable WiMAX and WiFi network in the CI Park at the California State University Channel Islands (CSUCI). The conducted tests have proven that the infrastructure indeed can sustain the data communication in remote areas, and that it can recover from natural power outages and continue data collection from the point of disruption.

Acknowledgements

I would like to thank my advisor AJ Bieszczad for his encouragement to join the MSCS program here. Also for his continued support for the project from conception to completion. If were not for his patience and long office hour talks this research would not exist.

TABLE OF CONTENT

TABLE OF FIGURES

Chapter 1: Problem Statement

1.1 Wireless Sensor Networks (WSNs)

A Wireless Sensor Network (WSN) is a network of small autonomous sensor nodes that continuously monitor their environment and report data to a centralized location. The collected data is then transmitted between other nodes on the local network, eventually arriving at a node connected to another network, such as the Internet. What this achieves for a researcher is similar to what the Cloud has achieved for everyday users: access to data at a location and device of the user's convenience. WSNs have a wide range of applications including monitoring: zebra migrations [1], building structure integrity [2], glacial movement [3], and light under shrub thickets [4]. A typical WSN and its common component is shown in Figure 1.

The wireless nature of the application allows researchers to place sensor nodes in an ideal position to collect optimal data without the necessary constraint of wiring that location. This has the advantage of not requiring a researcher to trek out to a remote location with the hope that the sensor has indeed been collecting data all along. There are cases where a sensor's expected lifetime may be unpredictable or may inevitably deteriorate over time due to a harsh environment. Since the sensor nodes are connected to the Internet, a sensor node may receive updates or report failures in real-time.

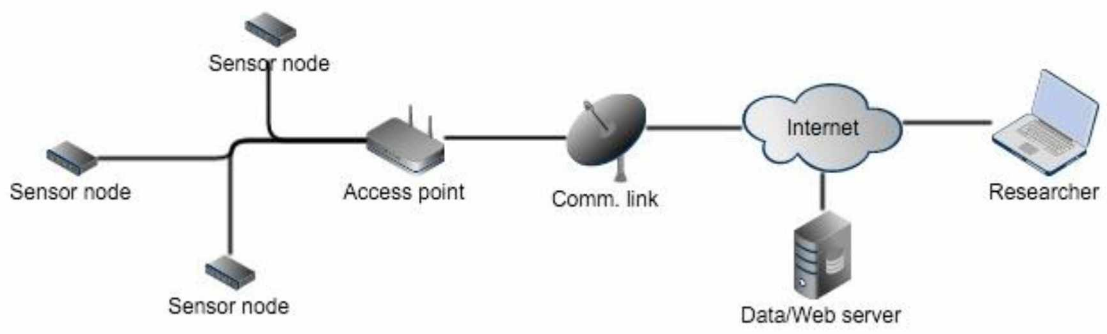


Figure 1. A typical WSN Layout

Several surveys have revealed that most WSNs are developed to collect data for a specific environment variable [5] [6]. The knowledge and effort required to deploy a system is substantial. The software alone involves the entire application stack from the hardware interacting embedded systems layer, the network protocol, to the end user level application layer. The hardware involved also provides many challenges depending on the environment and needs of the sensors.

1.2 The CI Rainbow System

In this thesis we describe the infrastructure and system for monitoring long-term environmental data in real-time with added managing capabilities along with tools to aid researchers in data analysis. The infrastructure provides a permanent and expandable network using wireless capabilities to connect sensor nodes over a long distance. The system itself provides a flexible platform for rapidly developing and deploying sensors to an existing infrastructure.

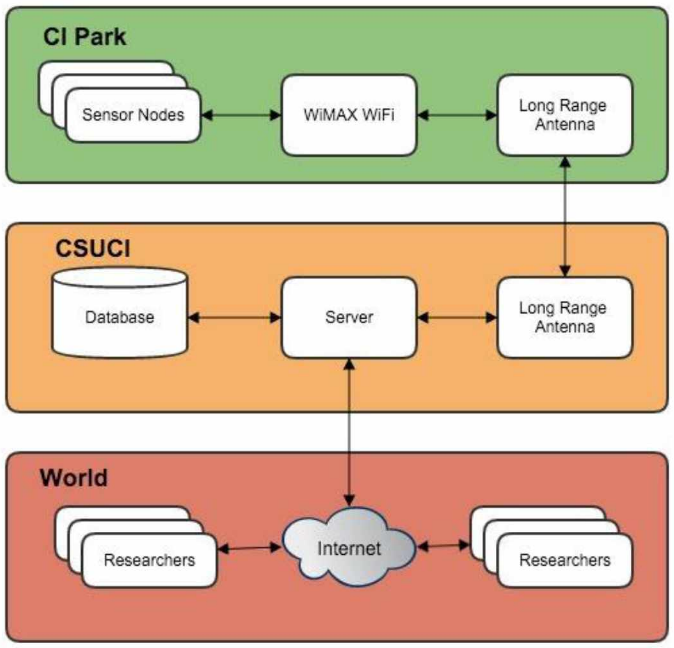


Figure 2. High Level Overview of CI Rainbow

The main features of the CI Rainbow software are:

Non-sensor specific design - We have developed a sensor node with the idea that the system will be employed by many different research departments, each with their own environmental sensors. To make the system flexible, sensors and their data are treated generically across the entire system.

Sensor node management - Due to the remoteness of Santa Rosa Island, the system may also be remotely managed through a web interface. A user may install, update, and change configuration files for a specific sensor through this interface.

Linux development environment – Since CI Rainbow will span across multiple departments and research leaders, we wanted to use a familiar development environment to make the creation and testing of sensors a less difficult task.

1.3 Actors

This section identifies and describes the actors involved in how CI Rainbow will be utilized in an academic setting.

Sensor Node

A sensor node is a collection of hardware and software whose primary purpose is to provide a flexible platform for collecting, storing, and transmitting sensor data. Sensor nodes run on a Debian based Linux operating system and perform their operations through the CI Rainbow node software.

Sensor

A sensor is a hardware and software element that collects data. The hardware is the sensor itself, along with any wiring required to send/receive data from the sensor node. The software is an entire package including any files necessary (configuration files, libraries, extra log files, etc.) to run.

Reading

A reading is a single unit of data collected from a sensor. The information that comprises is a reading is a unique node ID, unique sensor ID, date, and data from the sensor.

Event

An event represents a relationship between an action and a trigger. When a trigger is fired, the event will call an action to be performed. Events may also have titles and descriptions to convey to the user purpose of the event.

Action

The system supports two types of actions: server-side actions and node-side actions. Server-side actions take place on the web application server itself, such as sending an email or text message. While node-side actions require an action to be performed on a particular node. For instance, a user may want to call the camera sensor to take a picture if there is motion detected from another sensor on that node.

Trigger

A trigger is a sensor author defined integer that represents a data comparison check of some kind. For example, if a temperature sensor is reading above 100 degrees, a trigger may be thrown to the web application and any event using that trigger, will perform their respective action.

Web Application

The Web Application is responsible for many aspects. The web application must run a service for receiving readings, sending configuration changes, and registering sensor nodes. It also must provide the user with an interface for creating, editing, and viewing sensor nodes, sensors, readings, events, triggers and actions.

The web application also provides an interface for running a Jupyter notebook so that data mining techniques may be run on the data collected by the system.

Node Owner

The Node Owner are users who are the initial creator of the sensor nodes. Node owners are responsible for assigning a group to help manage the node and initially activating the sensor node with the system.

Group Member

Group members are users who may edit the sensor nodes and sensors that are associated with their group. They may also view any sensor nodes/nodes that are within that group.

Sensor Author

Sensor authors are users who create sensor packages for use in the system by node owners/group members. They are responsible for packaging the sensor in a tar.gz file, filling out the create sensor form on the web application, and define and describe any actions or triggers created.

1.4 Functional Requirements

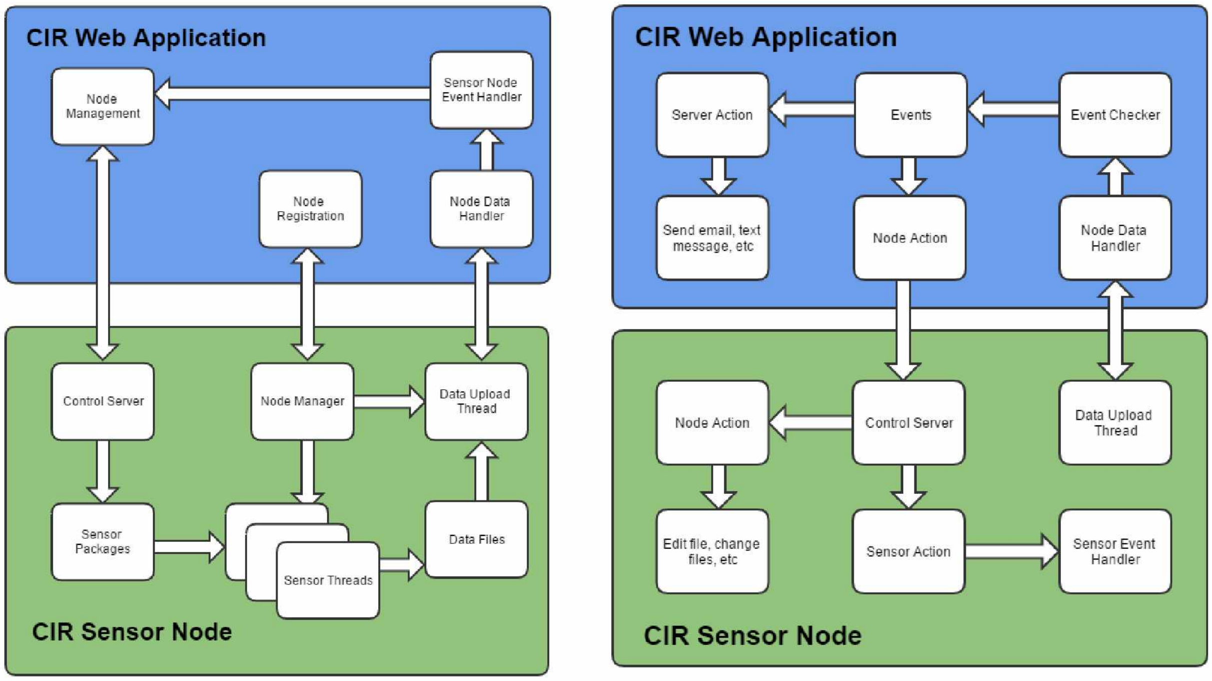


Figure 3. Interactions of the CI Rainbow Components

Create Nodes

Nodes are created by a user and associated with a group through a simple form interface on the Web Application. A name, user, and group become associated with the new node and a unique identifier is assigned. The node remains inactive until the setup script is run on the node hardware itself. The activation process is described in the Activate Nodes section.

Activating Nodes

Once a user has created a node, it will remain inactivate until the setup script is run on the node itself. An inactive node presents the user with a unique key and a link to the setup script to run on the node. This information is presented on the node viewing page of the Web Application. During testing, the sensor node can be configured to communicate with a testing server.

When the setup script runs on the node, the CI Rainbow Node Software is downloaded and installed on the node with preconfigured settings. The script will ask the user for the unique key given by the Web Application and then attempt to complete the process. If the key matches an inactive node, the node will be activated and ready to deploy.

Edit Node Information

The node information may be updated at any time by the owner or member of the group associated with the node. This is done through a form on the Web Application. If necessary, information may be transmitted to the Control Server.

Update Node Configuration Files

The ability to update node configuration files from the Web Application allows researchers to make remote changes to how a particular node is a running. The changes requested are processed through a form on the Web Application, translated to JSON, and passed to the Control Server running on the node. A secret key is required for all transactions between the Web Application and Control Server, this key is generated upon activation of the node and prevents unsolicited changes to the node's configuration files. Once the changes have been applied to the configuration files, the Node Manager receives a restart command to reload the files/sensors.

Create Sensor

Sensors are created by the user and associated with a group through a simple form interface on the Web Application. A name, description, group, and sensor package are required. The sensor package is a tar (tape archived) and compressed (gzip) file containing all the necessary files for the sensor to run.

Once a sensor has been created, any user may enable the sensor on their node through the sensor view page on the Web Application. Sending sensors to nodes is covered in the Enable Sensor section.

Enable Sensor

The node owner or group member of a node may enable any number of sensors on their nodes. Sensors may be enabled/disabled from either the node view page or the sensor view page. The sensor view page allows for the sensor to be enabled on multiple nodes in a single action while the node view page only a single node will receive the sensor.

Once the enable sensor request has been placed on the Web Application, the sensor package will be transferred to the Control Server along with the secret key agreed upon activation. This is to prevent unwarranted updates to the sensors. Configuration files may also be updated through the Control Server and the Node Manager will receive a restart command to load the new sensor.

Disable Sensor

The node owner or group member of a node may disable a sensor at any time through the Web Application. The command to disable the sensor will be sent from the Web Application to the Control Server along with the secret key.

Receive Data

The Web Application receives data from the Node Manager in JSON format including the unique node identifier, secret key, and data itself. The data is then stored in a MySQL database and may be used for analysis.

Create Event

Events may be created in the Web Application for use on a specified node. When creating an event for the node, the user is presented with a list of triggers and actions that can be associated with each other. The association of a specified trigger and action create an Event.

Register Action

Actions that are not server actions are created and defined by the Sensor Author. On the edit sensor view, the sensor author can register actions created with the Web Application through a form. The required information is an integer representing the Sensor action and a description of the action.

Register Trigger

Triggers are created and defined by the Sensor Author. On the sensor edit view, the sensor author can register triggers with the Web Application through a form. The required information is an integer representing the Sensor trigger and a description of the trigger.

Analyze Data

A python virtual environment is created and maintained for each user that wants to analyze data. This virtual environment is used for a separate instance of a Jupyter notebook running with any extensions required by that user. Data may be imported from the CI Rainbow Web Application to be used for analysis. Since each user has their own instance of a Jupyter notebook, sessions may be saved for future use.

1.5 Remaining Chapters

In the second chapter, we discuss previous research and recent advancements in the WSN field. We analyze the various software and hardware used in other previously developed systems.

In the third chapter, we fully discuss the implementation details of the entire CI Rainbow system. First we discuss the hardware used for the sensor nodes and network infrastructure, then we examine the core sensor node software system before looking at an example sensor package.

In the fourth chapter, we discuss the exploratory network deployed near the CSUCI campus at CI Park.

In the fifth chapter, we analyze the data collected about the sensor nodes, sensors, and system as a whole as it has been operating in CI Park.

The conclusion and future work about the research is presented in the sixth and seventh chapters.

Chapter 2: Field Overview

2.1 Preliminaries

Wireless Sensor Networks (WSN) have gained much popularity in recent years due in part to the inexpensive costs of the hardware and the multitude of potential applications [5]. The WSN market has grown significantly since 2010 and is forecast to continue growing through 2024 [7]. This growth includes both academic and commercial applications and is captured in Figure 4. The applications of WSN vary greatly and this has allowed for a great amount of research to be completed in the field. In an excellent and comprehensive survey [5], a range of the applications (Figure 5) and type of WSN are presented.

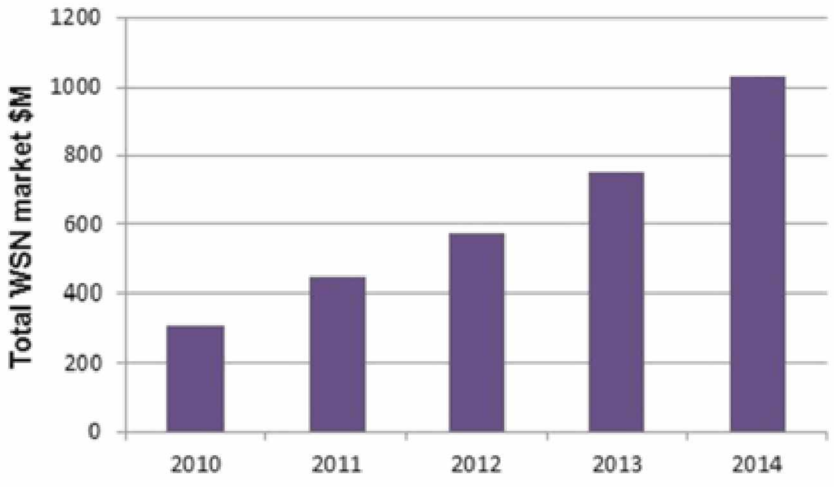


Figure 4. Growth of WSN Market [7]

A vast amount of research has been completed on improving many of the individual components of typical WSN. This includes clustering algorithms that improve scalability over distances [8], routing protocols to improve data transmission [9], time synchronization across sensor nodes [10], and security topics [11]. All of these components come together through hardware, software, and networking in order to establish a WSN.

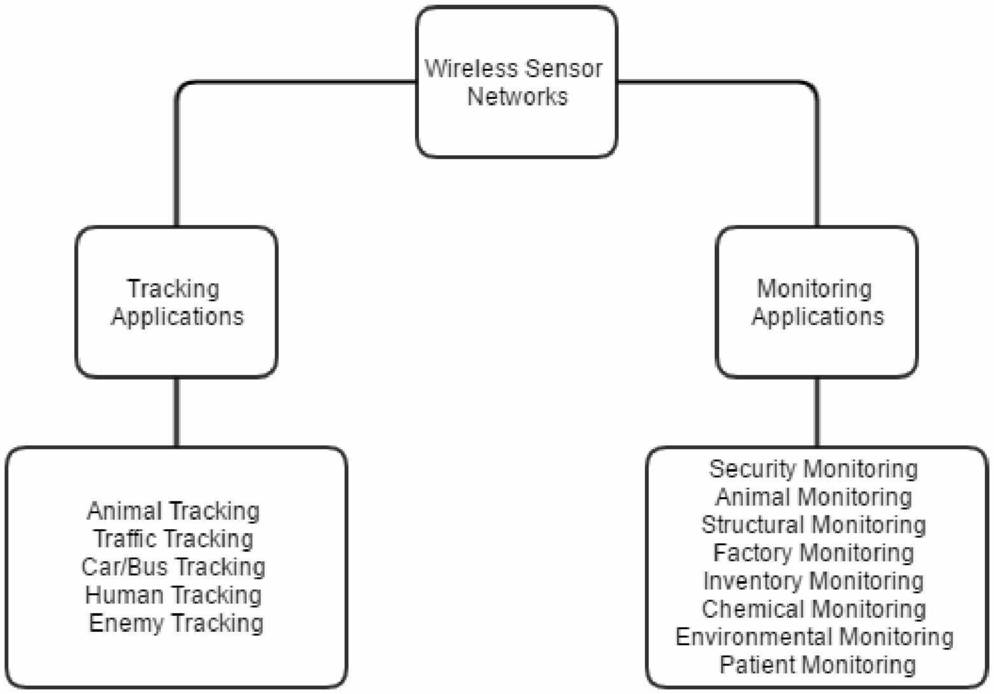


Figure 5. Applications of WSN [5]

2.2 Characteristics of WSN

WSN are deployed in remote areas and may contain many sensor nodes. The general characteristics and requirements for a WSN are energy efficiency, low-cost per sensor node, distributed sensing, wireless, and multihop [12].

Energy efficient – Sensor nodes should be energy efficient so that data may be collected over a long period of time without the need to revisit a node. Nodes may also rely on an additional power source such as solar to increase its lifetime. Since many nodes may be deployed on a single network, it would be infeasible to replace batteries on hundreds of nodes in the field.

Low cost – Each sensor node in the system should remain as affordable as possible since a network may consist of many nodes.

Distributed sensing – This provides robustness to collecting data in difficult environments where there may be obstructions or other obstacles.

Wireless – A WSN must be wireless in order to provide a method of transmission in an environment with little to no infrastructure.

Multi-hop – A sensor node's coverage may not include the data sink, therefore it is important for a node to use a multi-hop data transmission to transfer its collected data.

2.3 Components of WSN

A typical WSN is comprised of sensor nodes, sensors, data sink, transmission protocol.

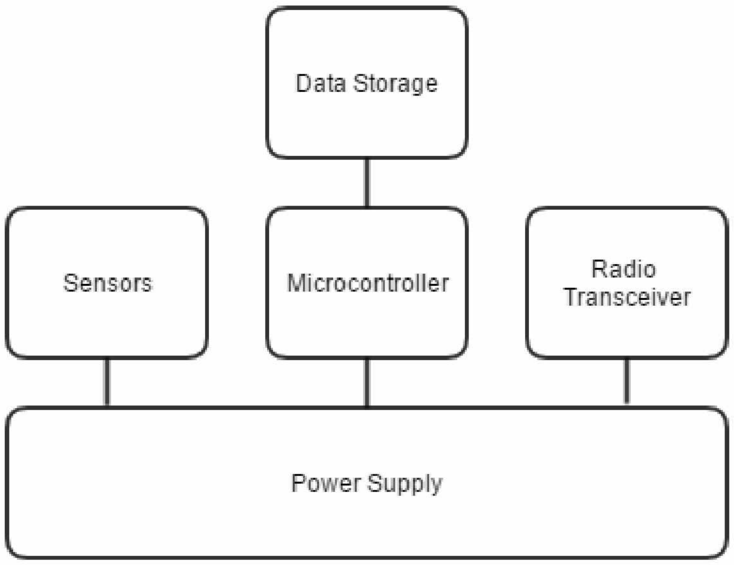


Figure 6. Architecture for a sensor node

Sensor node - A sensor node is the component responsible for some or all of the following: collecting data from single or multiple sensors, relaying data to a router, exchanging data with other networks, acting as a base station, or acting a sink node [13].

In terms of hardware, a sensor node is made up of a microcontroller, power supply, radio transceiver, and memory. The general architecture is shown in Figure 6. Each component of the sensor node can vary widely based on the needs of the WSN. For instance, a sensor node may be under heavy tree coverage and cannot rely on solar so a very small and efficient microcontroller must be used. Depending on the network infrastructure available, sensor nodes may communicate via WIFI, RFID, or another form of radio transmission.

For software, a popular choice for an operating system is TinyOS [14]. TinyOS is an operating system designed specifically for sensor nodes, and includes routines for common networking procedures that a sensor node may need to perform. Contiki is another option, providing many features along with dynamic module loading. Both of these operating systems are designed specifically for small and low power microcontrollers.

Sensor – A sensor is attached to a sensor node and is responsible for collecting data about its surrounding environment. Software must be written to communicate with each sensor specifically for the platform it is running on.

Data sink – A data sink is the location where data from a single or multiple nodes is transmitted to and stored before potentially being sent off to a router. Sensor nodes may also be data sinks themselves.

Router – A device that is connected to the Internet allowing for the transmission of data. A router may be a two-way satellite uplink device, WIFI router, or any other device connecting to an internet.

Infrastructure – Often WSN have little to no infrastructure [5]. Let us define the infrastructure as the physical and organizational structures that include the networking components of a WSN, allowing for data from the sensor nodes to reach an internet.

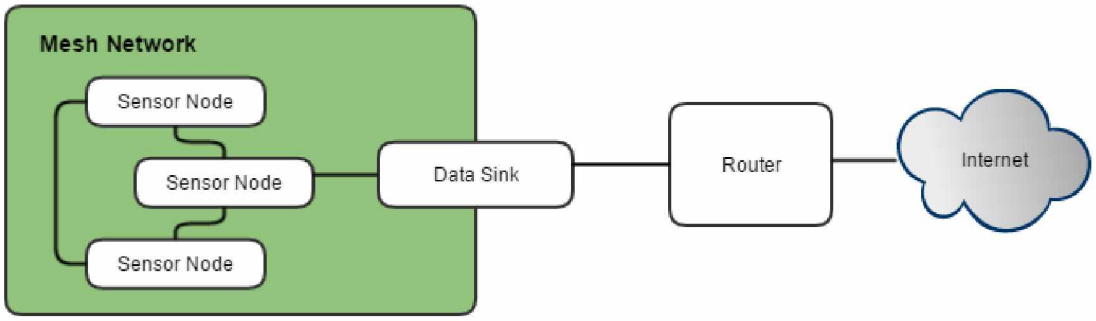


Figure 7. Mesh network layout example

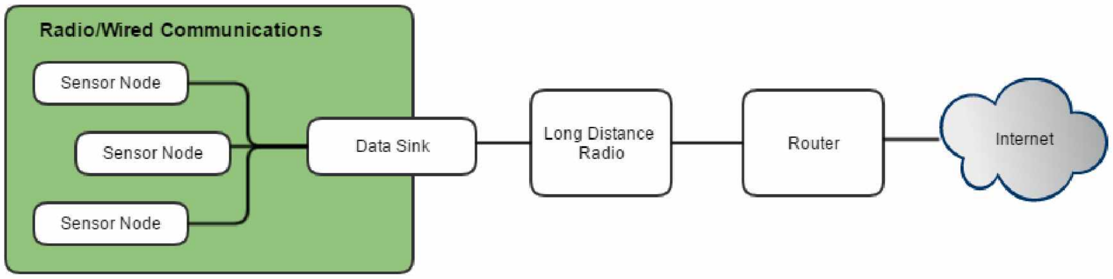


Figure 8. Radio communication layout example

The layout of a WSN depends on its requirements. Figure 7 shows an example of a mesh network or multi-hop network of sensor nodes. Data is collected and passed through the sensor nodes until reaching a data sink, where the data can be stored and transmitted to a router. Figure 8 shows a group of sensor nodes that communicate via wired or radio transmissions to a data sink. The data sink then uses a long distance radio to connect to a router.

2.4 Terrestrial WSN

WSN may be categorized into one of the following types: terrestrial, underground, underwater, multi-media, or mobile. Since the CI Rainbow system falls into the terrestrial category, we will focus on that type. A terrestrial WSN consists of a large number of sensor nodes deployed over land in a dense environment [6]. The remoteness of the environment adds to the limitations of the sensor nodes, which often required a secondary power source such as solar to continue operating. The open environment may create the need for a waterproof encasing, a distant and power consumptive source of transmission, and a requirement for minimal maintenance.

Collecting data via personnel can be difficult if the location is remote and even potentially dangerous. The logistics and cost of sending researchers out to collect data will also increase over time. A terrestrial WSN collecting environmental data is ideal as there is a small initial deployment cost and only followed occasionally by some maintenance.

Great Duck Island

The Great Duck Island (GDI) WSN is a deployed network of 32 nodes on a small island 15 km off the coast of Maine [15]. The goal of this WSN is to collect data for biologist to monitor the seabird nesting environment and behavior. Sensor nodes were developed on the Mica hardware platform, the largest UC Berkely mote available at the time. The nodes are powered by a pair of AA batteries, providing enough power for six months of deployment. Each sensor node was equipped with a temperature, photoresistor, barometric pressure, humidity, and thermopile sensors. A gateway node ran on a larger microcontroller with a Linux OS. These gateway nodes provided internet connectivity for a patch network of sensor nodes within the area. If a sensor node was out of range of the gateway node, it would forward its messages to the other nodes in the area to eventually arrive at the gateway.

Redwood Tree WSN

The Redwood Tree WSN [16] was deployed to a single 70 meter tall redwood tree in Sonoma, California. A total of 30 nodes were deployed at various heights and radius distance on the tree. The goal of this WSN was to observe the microclimate over an entire redwood tree, as there is substantial variation in readings and temporal dynamics. The WSN was deployed for 44 days, and collected data at 5 minute intervals. The sensor nodes used were Berkeley Motes, specifically the Mica2Dot by Crossbow. Each sensor node was equipped with temperature, humidity, and light levels (both direct and ambient). A gateway node was also deployed to receive data from the sensor nodes.

Glacsweb

Glacsweb is a WSN deployed on a glacier in Norway in 2003 [3]. Nine probes connected to a sensor node collected data that was transmitted from a base station to reference station 2.5 kilometers away. The system was designed measure the relative motion of the glacier bed to the surface, the motion of small rocks in the bed, and other parameters for glacial dynamics. The sensor nodes are attached with the following sensors: pressure, temperature, orientation, external conductivity, and strain gauges. Data is collected six times a day, and transmitted once to the base station. The base station is run on a larger ARM based microcontroller with a Linux OS. The choice to run Linux was made in order to have better remote access for debugging and maintenance.

LUSTER

LUSTER is the Light Under Shrub Thicket for Environmental Research: a WSN designed to monitor the effects of sunlight for shrub thickets [4]. This WSN has been deployed on Hog Island off the Eastern Shore of Virginia. Transmission of the data from the island to the mainland is accomplished by a long-distance wireless data link. The data collected from this system is displayed through a customizable web application view. With the web view a researcher can analyze data in near real-time, since there is some delay between transmissions from the sensor nodes.

Environmental Monitoring Application

The Environmental Monitoring Application (EMA) WSN has had multiple successful deployments [14]. One deployment, discussed in [17], is an urban WSN deployment of 15 sensor nodes over a 12-month period. The goal of this project was to design a maintenance free WSN for collecting environmental data. The sensor nodes were attached with temperature and relative humidity sensors. The nodes were then deployed within a close proximity on lamp posts in an urban environment using a multi-hop transmission technique to send data to the data sink. TinyOS was selected to run on the sensor nodes due to its popularity, active development, and feature rich abilities.

2.5 Challenges and Difficulties

In reviewing surveys and applications of WSN [18] [4] [19] [3], we list the most common challenges and difficulties these systems have encountered in this section.

Power management – This is important for WSN deployed in remote locations or hostile environments. In most cases, there is no supporting infrastructure to continuously provide power so it is essential for a sensor node to manage its energy budget efficiently. There are also instances, such as with Glacsweb or GDI, that the sensor node is placed underground and cannot reach direct sunlight rendering solar solutions infeasible.

Management and usability – Deploying a WSN requires a considerable amount of technical expertise ranging in both hardware and software. The research platforms discussed in the previous section are not easy to maintain, and understand by average technical users. They also are not necessarily easily adaptable to sense different environmental variables as well.

IP end-to-end connectivity – By incorporating this feature into sensor nodes, the management, configuration, and debugging process becomes much simpler for developers. By having a sensor node directly connect to the Internet, data can be accessed in real time by researchers.

Data mining – Developers should put forth effort into making data mining simple for users and researchers. Standardizing the data format and how the data can be retrieved by researchers will greatly aid this effort.

Delay tolerance – The reliability of the network or power available in a harsh environment may vary day to day, so WSN must be designed with delay tolerance in mind. If data cannot be transmitted for a given time, the sensor node must store it temporarily until a transmission window opens again.

Encapsulation – Sensor nodes contain sensitive electronics that must be protected from extreme heat, moisture, and other harsh environments. Protecting these components while also not impeding on the sensors ability to accurately detect its environment is an issue encountered on almost every WSN.

Security – Since sensor nodes are running on smaller microcontrollers, they may not have the processing power necessary to enforce conventional security methods. It is also possible that a sensor node does not have the energy budget to provide the security required.

Standardization – The IEEE 802.15.4 lays out specifics for low-rate wireless personal area networks (LR-WPANs). The standard was introduced in 2003 and was designed for applications that need to transport smaller data samples with limited power and resources.

Chapter 3: Technical Details of the Work

3.1 Overview

CI Rainbow is a sustainable terrestrial WSN currently deployed in CI Park. In this chapter, we describe the infrastructure supporting the WSN, the hardware for sensor nodes and sensors, the software managing the sensor nodes, the web application software, and the data mining software.

The general use case supported by this system is a researcher who wants to develop a sensor and deploy a sensor node to the network. This task is accomplished through the following steps:

- Create a sensor node through the web application.
- Initialize the sensor node hardware with an automated script.
- Develop and upload a sensor package to the web application.
- Download the sensor package to the sensor node via a script on the sensor node.
- Test the hardware and software together.
- Activate the sensor node (flag as ready for deployment).
- Deploy the sensor node into the CI Rainbow network.
- View the data in near real time through the web application.
- Download and perform data mining techniques through Jupyter notebook.

3.2 Infrastructure

The current infrastructure deployed across the CSUCI campus is a proving ground for a future deployment to Santa Rosa Island. This section describes the geography, environment and the main components of the infrastructure. The main parts of the infrastructure are:

- Exploratory wireless network.
- Application and data center.
- Sensor node platform.

CI Park

The development of the CI Rainbow WSN is motivated by the recent opening of the California State University Channel Islands Santa Rosa Island Research Station (SRIRS). Santa Rosa Island is one of the Channel Islands located approximately fifty miles west from California State University Channel Islands (CSUCI). There is already data being collected on the island by several researchers at the university, but this data is manually recorded as of this writing. The ultimate goal is to deploy the CI Rainbow WSN on Santa Rosa Island. However, due to the remoteness, protective status, and lack of utilities developing a system on the island is not currently feasible. Instead, we chose to deploy an exploratory infrastructure in a wildlife park owned by CSUCI and neighboring the campus. The infrastructure is shown in the figure below. This park is nestled in the northern tip of the Santa Monica Mountain range and possesses many features of which Santa Rosa Island has:

- Mountains and valleys.
- Similar environmental conditions.
- Similar plant diversity.
- Lack of utilities, such as power and Internet access.

Exploratory Wireless Network

The exploratory wireless network provides an internet, referred to as the CI Rainbow network, for the sensor nodes and web application. Each router in the infrastructure is set to bridge mode allowing the entire system to be treated as a single internet.

Figure 9 shows the physical layout of the infrastructure with images of the various nodes at their location. We use Ubiquity WiMAX equipment to communicate over long distances. The long distance antennae require a line of sight so to overcome the mountains between the campus and CI park we must use several relay nodes to transfer the data. The first one relay is near a water tank, hence the label “water tank.” From this vantage point, the “ridge” relay node is visible and the ridge area overlooks the entire CI park. So from this ridge, we have the ability to connect to three CI Park sensor platforms placed throughout the park.

This network differs from a typical WSN in that it heavily relies on the infrastructure in place. An obvious disadvantage is that to add sensor nodes to a new location requires a line of sight to one of the long range relay nodes. However, it allows us to have a connected WSN, meaning we have the capability of communicating in near real time with sensor nodes anywhere with Internet.

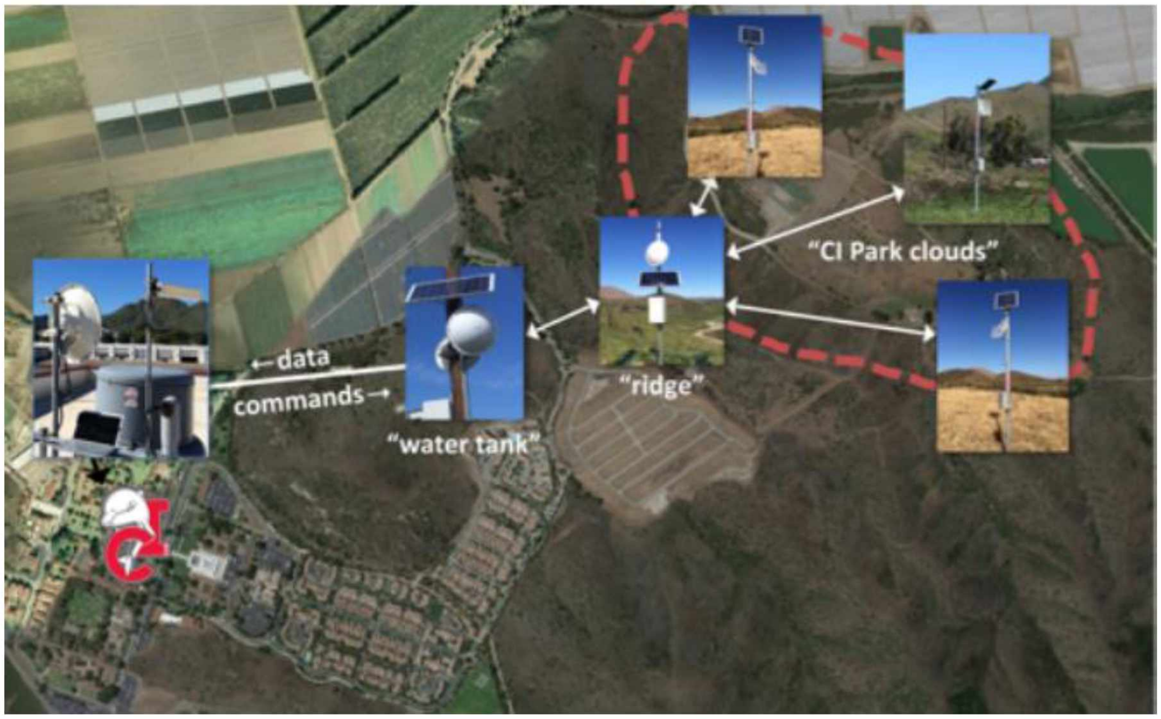


Figure 9. The exploratory wireless network in CI Park.

Application and Data Center

The application and data center is located inside the CSUCI data center on campus. A single server running as a virtual machine host runs all the virtual servers required to serve the web application. There are two virtualized servers running: the web application server and the database server. The servers are virtualized using VMWare and may be managed remotely for convenience.

The server hardware consists of a Dell PowerEdge R420 with 2 x 1.6 GHz Xeon CPUs, 15 MB cache, 32 GB RAM, and RAID 5 with 4 x 4 TB hard drives. This provides plenty of power and bandwidth to currently host the project in its research state. The web application server has a connection to both the outside network (the Internet) and the CI Rainbow private network since the web application interacts with both users and the sensor nodes. The web application and database server both run Ubuntu 16.04 LTS as their operation system.

Long Range Relay Nodes

The large distances and environment between the data center and the sensor nodes require long range relay nodes to connect them. Since mountains are obstructing the view of the sensor platforms from line of a sight, several relay nodes have been placed between the campus and CI park.

These relay nodes are solar powered by 100 Watt panels which store power in two large batteries in series. Each node has relay things and line of sight stuff. These are placed on high vantage points in order to minimize the number of relay nodes required for the system.

Sensor Node Platform

The sensor node platforms create WIFI access points allowing the sensor nodes to connect to the CI Rainbow network. For convenience to researchers and to avoid conflicts, the WIFI routers allocate addresses dynamically through a DHCP server. For security, each access point uses WPA2 authentication, so they cannot be accessed by regular WIFI clients.

3.3 Sensor Node Hardware

In this section we describe the hardware components used for sensor nodes and select sensors. Many of the sensors were assembled by undergraduate students as a senior project.

Sensor Node

We have chosen the raspberry pi as the microcontroller for sensor nodes because they are widely available and are capable of running a Linux operating system. The Raspberry Pi is available in many different models, we have chosen the smallest and cheapest version. The Raspberry Pi Zero is a small microcontroller capable of running a Linux operating system. It is equipped with the following:

- Single core 1 GHz ARM CPU
- 512 MB of LPDDR2 SDRAM
- MicroSD card slot
- Mini-HDMI output
- Micro USB input
- Micro USB power
- 40-pin GPIO header
- 65mm x 30mm x 5mm dimensions

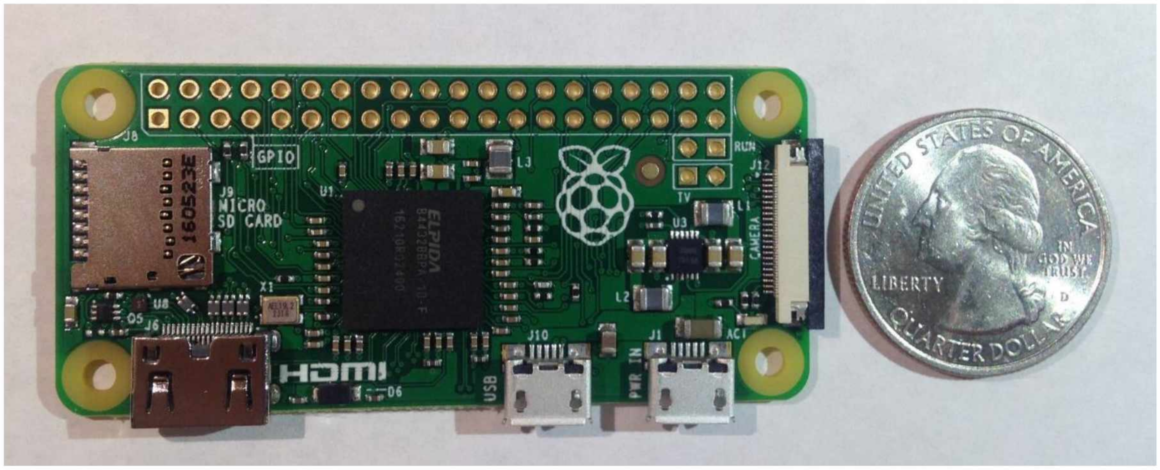


Figure 10. Raspberry Pi Zero with a US quarter for reference.

These microcontrollers are widely available and cost 5 dollars per unit. Additional costs are required for the SD card and the cabling, however, this only brings the total cost to 20 dollars per unit.

3.4 Node Software

This section describes the software environment and software that runs on sensor nodes.

Operating System

We use the Raspbian Linux OS, which is based off of Debian and is the official operating system for the Raspberry Pi. We have chosen to use a Linux OS rather than the popular choice of TinyOS or Contiki since our microcontroller is larger and more powerful. There are several benefits to this choice: Linux is a familiar development environment for users, Raspbian can be emulated on QEMU, the availability of a wide range of open source tools, and a variety of programming languages are supported.

Since Raspbian is the officially supported OS for the Raspberry Pi, no additional customization is required to run the CI Rainbow system. Some packages need to be installed in order for the system to run, but that is handled by the system installer.

Initialize Script

The initialize script is responsible for initializing the system for use in CI Rainbow. This script was written in Python3 and configures the following:

- *Environment variables* – Specifically the CIRAINBOWPATH variable that contains the root path of the CI Rainbow system.
- *Required libraries* – The system requires a few extra libraries not included in the default Raspbian installation. The script will check if these exist on the system, if not, it will install them.
- *CI Rainbow server information* – If a custom CI Rainbow web application server is required, the user may enter a server to be used.
- *CI Rainbow system* – The script will download the latest CI Rainbow system package from the web application server and install it to the CIRAINBOWPATH directory.

Sensor Install Script

Once a sensor node has been initialized and activated in the system, the user may run the sensor install script to download and install sensor packages from the web application. Although sensors may be installed through the web application remotely after deployment, it is important for researchers to have the ability to install sensors before the node is attached to the CI Rainbow network. When the script is run, it will prompt the user for a sensor ID, confirm, and then proceed to download and install the sensor package (if it exists).

The researcher may now run tests with the node manager and sensors to ensure everything is in working order. However, until the node manager is activated, it will not send data to the web application. Data will be collected from the sensors, but never uploaded during the testing phase. Once the node is activated through the web application, then data shall be uploaded.

Directory Organization

All CI Rainbow software, including sensor software, is placed in a root directory described in the environment variable CIRAINBOWPATH. The organization of the system borrows from a typical Linux folder structure.

- `/bin` – All non-administrative executables are stored in this directory. The node manager will only scan this directory for sensor executables.
- `/conf` – This directory contains additional configuration files necessary for the system.
- `/data` – All data files, files with the ".dat" extension, are stored here. The node manager will queue all files here for each send interval. Each sensor should only place data in this folder.
- `/data/queued` – The files contained in this directory are queued and are ready to be sent to the web application. Once sent, they are archived until deleted by a cron job. If the node manager fails to send the data, they remain in the queued directory.
- `/data/media` – Media files, such as images or videos, are placed by sensors in this folder. Due to the potentially large files sizes, media files must be handled differently by the node manager, so they are stored here.
- `/log` – Log files for the node manager, control server, and sensors are stored here.
- `/sbin` – Administrative executables are placed in this directory. This includes the node manager and the control server.

Configuration Files

The configuration files are required to be in a format similar to the INI file format (see Figure 11). The format allows for key and value pairs to be placed in sections and the ConfigParser Python3 module handles reading/writing to this format. By enforcing a single format for the configuration files, the web application can send update commands for any configuration file in the CI Rainbow system. One benefit of this is the ability to change a sensor's configuration file on demand through the web application.

```
[sensors]
  temperature=19,temp.py,300
  humidity=20,humidity.py,600
```

Figure 11. Example of the configuration format.

The main configuration file for the CI Rainbow system is named "cirainbow.conf" and is located in the `/conf` directory. This file contains the node ID, the IP address of the node, the data upload interval, and a list of the sensors. Sensors must be described in the "sensors" section with the following information: an all lowercase identifier with no spaces as the key, and the value is a comma separated string containing the sensor ID, sensor main script/binary, and the polling interval in seconds. Figure 11 shows two examples of sensors on a node, one of them being a temperature sensor with the ID of 19, main sensor script temp.py, and a 300 polling interval.

Data Files

All completed data files, including media, are stored in the `/data` directory. The node manager will scan this directory for any files and upload them to the web application for processing. Data must be packaged in the JSON format so that all data in the system is in a standardized format (see Figure 12). The format is specified as follows:

- A JSON object with two strings, date and data.
- The date string has an integer value representing the current date in the UNIX time stamp (seconds since Thursday, January 1, 1970).
- The data is a JSON object containing the data however the sensor author prefers to include the data.

```
{'date':546002400,'data':{'temp':72.3,'humd':20.2}}
```

Figure 12. Example of data in JSON format

By providing a standard format for all data in the system, using data from different sensors and research groups becomes much easier. Another benefit of using JSON is the native support Python has for handling the format. Since the data mining tools provided by the web application run on Python, data can be imported through a simpler process.

Node Manager

The node manager is a Python3 script that responsible for the following:

- Reading the configuration files and determining which sensors to run.
- Starting timer threads for polling the sensors.
- Starting a timer thread for sending data to the web application.
- Starting the control server.
- Queueing and uploading data to the web application.
- Waiting for restart command.

The node manager begins by reading the main configuration file, which contains all the sensor information and which sensors to load. The configuration files are described in the previous section. Once the system variables are loaded, the node manager starts timer threads for each sensor, and a timer thread for the data upload process. Now the node manager sleeps and waits for an action to perform. This process is shown in Figure 13.

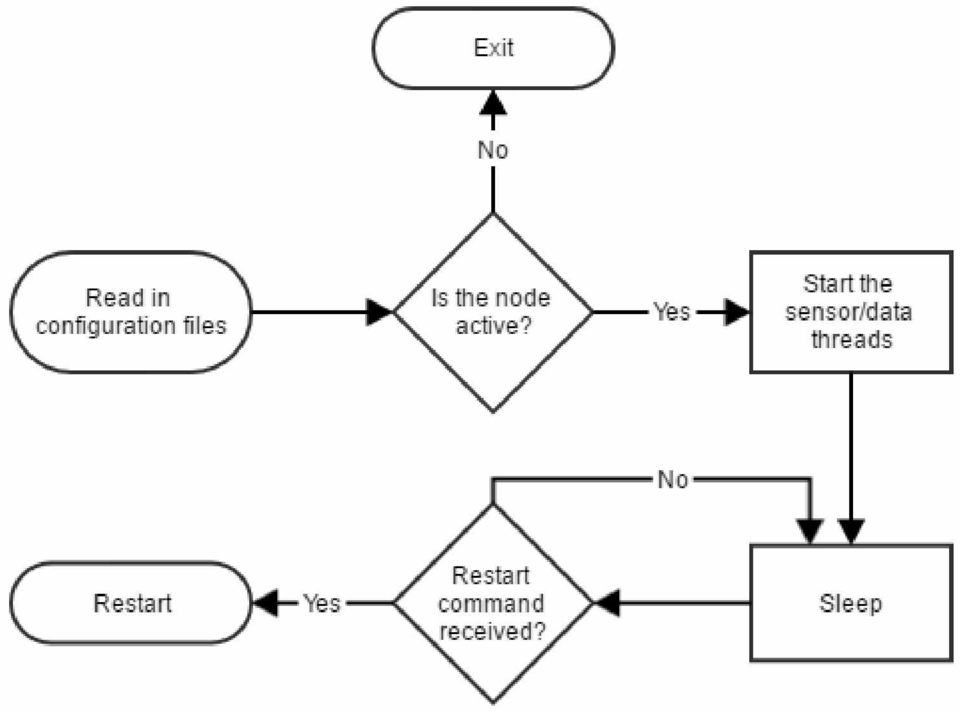

Exit
No
Read in configuration files
Is the node active?
Yes
Start the sensor/data threads
No
Restart
Yes
Restart command received?
Sleep

Figure 13. Flowchart describing the node managers process

From the sleep state, the node manager is waiting for three events: restart, poll sensor, and upload data. If a restart command is received, the script will restart itself and begin the process again. A restart command is generally issued by the control server after it has updated the configuration files. The node manager watches a hidden file called ".node_running", which contains the timestamp of when it was started. If this timestamp is updated by the control server touching it, the node manager will restart.

When the poll sensor timer expires, the node manager forks a new process on that thread to poll the sensor. Once the sensor has completed its task, the timer is reset and the thread's current state becomes wait.

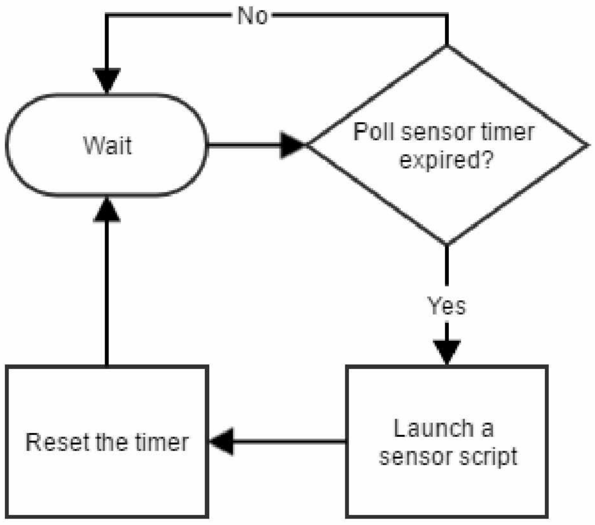


Figure 14. Node manager's poll sensor action flowchart

When the upload data timer expires, the node manager searches the data directory and moves the data to the queued directory. From here, it will attempt to upload the data to the web application and archive the data if successful. If the upload fails, the data remains in the queued folder and will try again at a later time. This process is described in the flowchart in Figure 15.

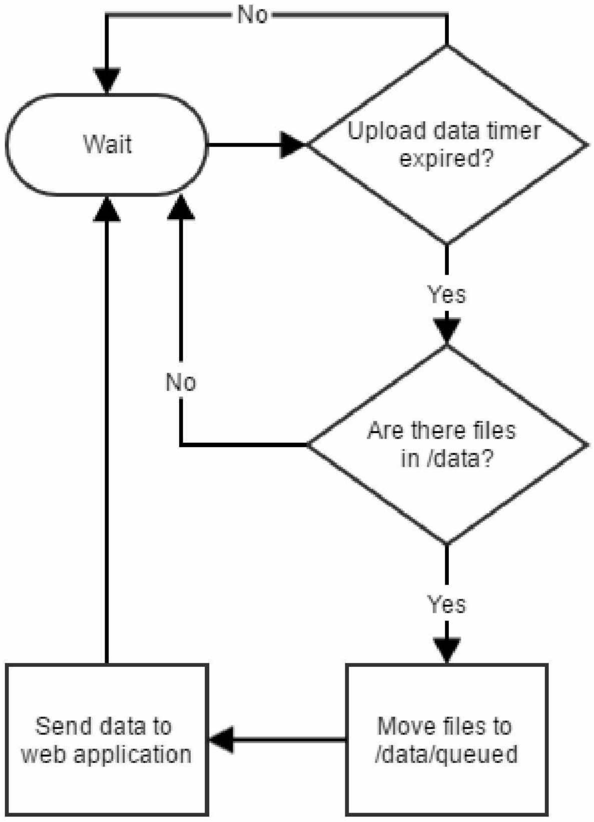

No
Wait
Upload data timer expired?
Yes
No
Are there files in /data?
Yes
Send data to web application
Move files to /data/queued

Figure 15. Flowchart describing the upload data process.

Control Server

The control server handles all remote management requests and was written in Python3. It provides a REST API over HTTP on port 9001 and listens for commands from the web application. For security, a previously agreed upon secret token is given for each command sent. There are three supported commands: update a configuration file, update a sensor package, and restart the node manager.

Purpose	**Type**	**URL**	**Parameters**
Restart node manager	POST	/restart_manager	token: A secret token to ensure commands are coming from a known source.
Update configuration file	POST	/update_config	commands: A JSON string of configuration commands. token: A secret token to ensure commands are coming from a known source.

Update sensor package	POST	/update_sensor	snapshot: A tape archived and gzipped package containing the sensor files. token: A secret token to ensure commands are coming from a known source.

Figure 16. REST API description for the control server.

When a restart manager command is received, the control server touches the ".node_manager" file to update its timestamp. The node manager will wake from its sleep state, read the change, and then perform a restart. A restart may be desired after updating a sensor or configuration file. If any errors are encountered, a JSON string is returned to the web application.

The update configuration command receives a POST parameter called "commands." This parameter contains a JSON array of JSON objects with configuration commands to execute. The options for the objects are described in Figure 17. The order of execution for each command is the same as their order in the array.

Command Name	Command Parameters	Purpose
add_section	section	Adds a new section, if the section does not already exist.
remove_section	section	Removes the section, if it exists.
edit_section	section, new_name	Renames the section to new_name.
add_key	section, key, value	Add a key and value pair to a section.
remove_key	section, key	Removes a key from a section.
edit_key	section, key, new_name, value	Updates a key from a section to with either new_name, value, or both.

Figure 17. Table describing the command options for the control server

The update sensor command receives a snapshot of the sensor through the POST parameter "snapshot." This file is downloaded to the root of the CI Rainbow system and unpacked, overwriting and adding the necessary files. The server responds with a JSON message with "success" or "failure" and an explanation of the failure.

3.5 Server Software

This section describes the software environment and software for the server component of CI Rainbow.

Component Overview

The main components of the CI Rainbow web application are spread over two servers. One server runs the web application and Jupyter notebook instances and another runs the MySQL database. It is a common practice to segregate the web application and database so we have done so as well. Figure 18 shows the core components and their interactions. Each component is covered in the next sections, however, Jupyter notebooks are discussed in 3.6.

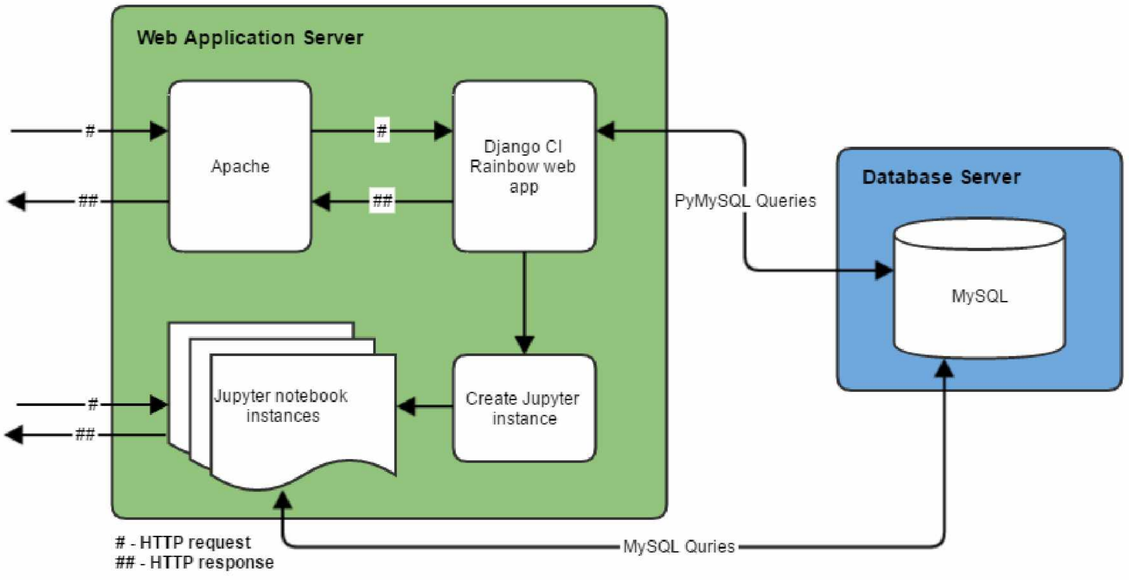


Figure 18. Server software interactions

Apache

In order to provide a robust system for a web application we chose to use the Apache2 2.4.18 web server. We have configured Apache to forward HTTP requests to the Django service using the Web Server Gateway Interface (WSGI) from the apache mod, “mod_wsgi.”

Django

For the web application, we have chosen the Django 1.10 python framework. This provided the necessary features to rapidly develop a secure web application that is portable and maintainable. The most useful features Django provides are: automatically generated database models based on object classes, regular expression URL matching, built-in user and group management, user authentication with salted passwords, testing platform, and model-view-controller style interactions.

MySQL

MySQL is not officially supported through Python3 and Django, however, it may still be used efficiently through the PyMySQL module. By default, Django uses SQLite, which does not fit our needs. Since we are running the database on a separate server, SQLite cannot be used as it does not have networking support. The database is automatically generated from the classes in the “models.py” file. All attributes and relationships are defined as a Python object, which allows us to manipulate the data as Python objects without any conversions. This also makes the application easily portable between database choices, as we can change the database connector at any moment.

REST Architecture

Representational State Transfer (REST) is an architectural style for distributed media systems [20]. Communication is stateless and does not rely on any server side stored information for requests. This has the advantage of improving scalability since the server does not have to manage resources. The stateless nature also helps with recovery from partial failures. When the architecture is applied to a web service, it is often referred to as a RESTful API. The following must be defined for a given RESTful API interaction: URL, media type (text, JSON, comma separated values, etc.), and a standard HTTP method (GET, POST, DELETE, etc.).

We have chosen to use a RESTful API for the communication between sensor nodes and the web application as it provides a simple uniform interface that can be easily expanded. Scalability is also important to allow for the system to accommodate a potentially large number of sensor nodes.

CI Rainbow Web Application

The CI Rainbow Web Application is a Django application written in Python3. The web application serves two purposes: provides an interface for users and sensor nodes. For users, the web application provides the ability to create, read, update, and delete (CRUD) the models. For sensor nodes, the web application provides a REST API for uploading data and media files.

The architecture of the web application follows to MVC (Model, View, and Controller) style. Models are objects that contain information about an item in the system. The view is the output representation of the model. The controller chooses a model to be used. In terms of a web application, when a user navigates to a URL, the backend of the application calls a controller, which manipulates a model with information from a database, and returns an HTML view containing information from the model.

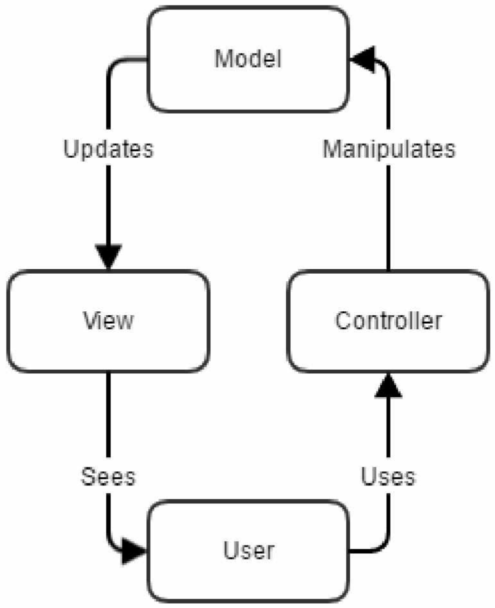


Figure 19. Model, view, and controller style.

Sensor Node REST Interface

The RESTful interface for the sensor nodes receives readings from the nodes and activation requests. At the time of activation, a secret token is generated and is used for communication between the nodes and the web application. This token is attached as a parameter to each request in order to ensure that data is coming from the node.

Purpose	**Type**	**URL**	**Parameters**
A new node must send an activation request to the web application and establish a secret token.	POST	/node/activate	node_id: The assigned node id from the web application. Returns a secret token upon successful activation.
Data from the sensors are sent to this URL for processing and storage.	POST	/reading/add	node_id: The node id. sensor_id: The ID of the sensor that the reading is from. data: A string of JSON data as described in the previous section.

			Returns a "success" or "failure" message.
Larger files require a different method of transmission. Generally, these files are media (images, videos, sound).	POST	/reading/add_media	node_id: The node ID. sensor_id: The ID of the sensor the reading is from. media: The media file or files (large files). Returns a "success" or "failure" message.

Figure 20. Table describing the RESTful API for sensor nodes.

Web Application Models

The models used in this system are described in the "models.py" file. Each model is a Python class with defined relationships and attributes. We defined the following models:

- Node – This model represents a single sensor node in the system. A node is defined as a name, owner, group, IP address, latitude and longitude, and an active flag.
- Sensor – The sensor model represents a single sensor in the system. We define this as a name, description, owner, group, and a sensor package.
- Reading – The reading model represents a single data point. Any data collected from a sensor is considered a reading. A reading is defined as a string of JSON data, a node ID, and a sensor ID.
- Event – An event model is a relationship between a trigger and action. An event is defined as a name, description, trigger ID, and action ID.
- Trigger – The trigger model represents a sensor author defined integer (code) representing a potential sensor trigger. A trigger is defined as a name, description, code, and sensor ID.
- Action – The action model represents a particular server action to be performed by an event. An action is defined as a name, description, code, and action ID.

Django automatically generates the database based on our models. The schema is shown in Figure 21.

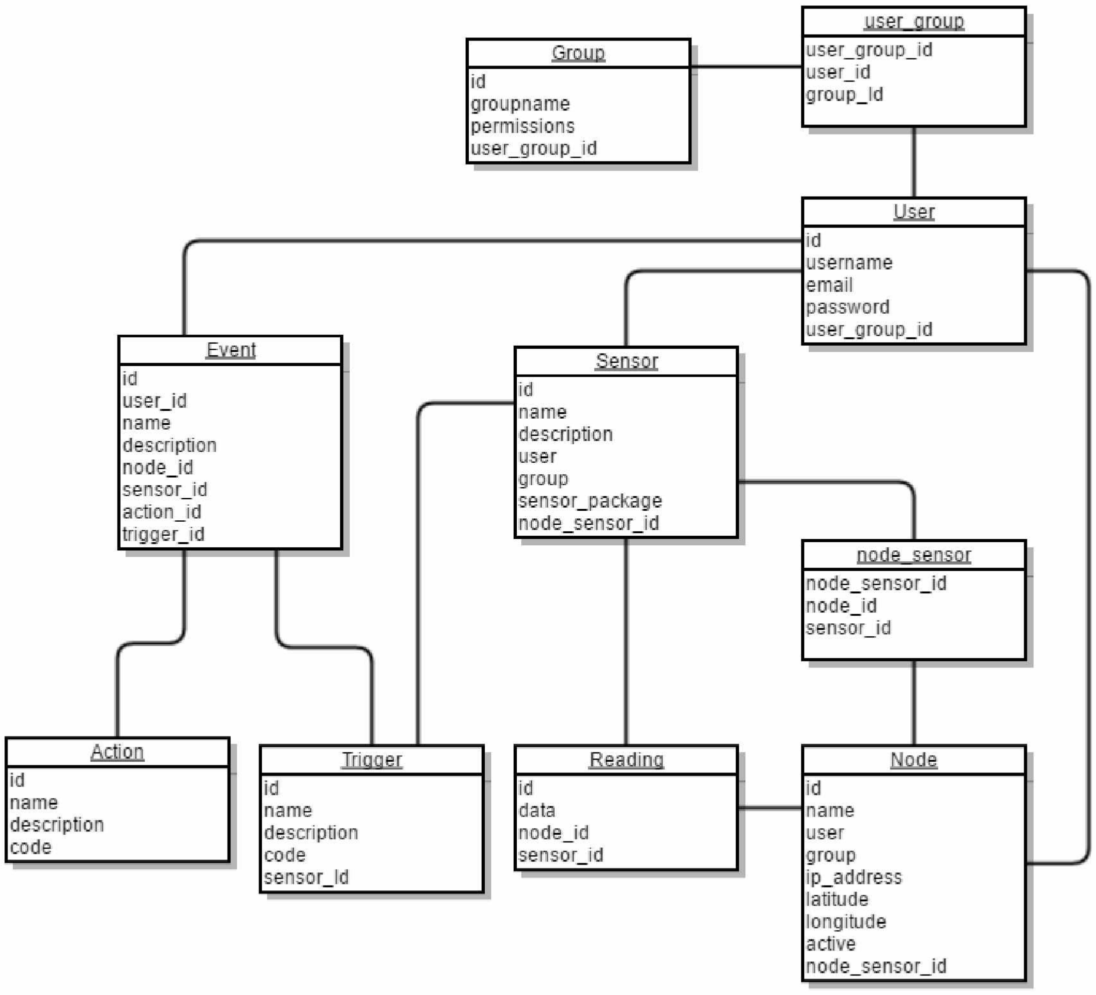
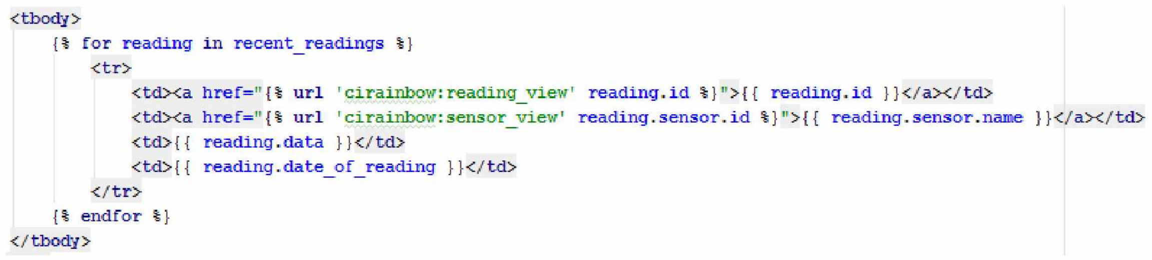


Figure 21. Database schema for CI Rainbow

Web Application Views

A view, in terms of a web service, is an HTML page displaying some kind of data. To present data on the HTML page dynamically, we take advantage of Django's build in tag system and custom tag system. Built in tags perform operations such as displaying data from a python object, formatting dates, control flow, or displaying forms. Some pages require custom tags in order to display more complicated data. An example of a template with tags is shown in Figure 22.



```
<tbody>
    {% for reading in recent_readings %}
        <tr>
            <td><a href="{% url 'cirainbow:reading_view' reading.id %}">{{ reading.id }}</a></td>
            <td><a href="{% url 'cirainbow:sensor_view' reading.sensor.id %}">{{ reading.sensor.name }}</a></td>
            <td>{{ reading.data }}</td>
            <td>{{ reading.date_of_reading }}</td>
        </tr>
    {% endfor %}
</tbody>
```

Figure 22. Example of an HTML template with Django tags

In order to reuse similar code, we place views in templates. These templates may be included into other templates. For instance, since the header and footer of the view is the same for all pages, the code for these is placed in its own template and included in each view. We follow the CRUD (Create, Read, Update, and Delete) paradigm for views, so some templates are reused for all the models. There are two views used to display common operations:

- Simple list – A template used to list nodes, sensors, readings, events, actions, and triggers.
- Simple edit – A template to display a Django form. This is used for node, sensors, events, actions, and triggers.

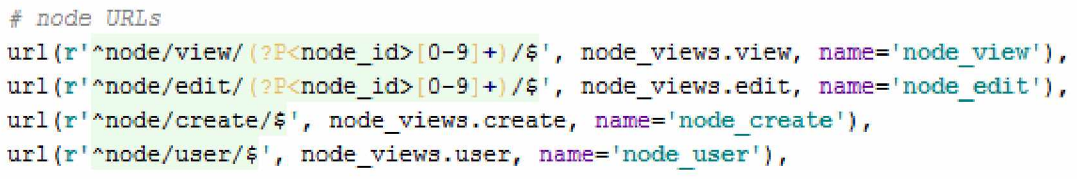
The views we have for our models are described below:

URL	**Purpose**
node/view/<id>	View the node with the ID of <id>, a list of sensors enabled, and recent readings received from the node.
node/edit/<id>	If the authenticated user owns the node or is part of the node's group, display an edit form for the node of <id>.
node/create	Create a node to be added to the system.
sensor/view/<id>	View the sensor with the ID of <id>, a list of nodes the sensor is enabled on, and recent readings received by the sensor.
sensor/edit/<id>	If the authenticated user owns the sensor or is part of the sensor's group, display an edit form for the sensor of <id>.
sensor/create	Creates a sensor to be used in the system.
reading/view	View a single data point collected by a particular sensor and node.

Figure 23. Table of the various views and their purposes.

Web Application Controllers

The URLs from the previous section each have their own controller which manipulates a model to be viewed. URLs and controllers are paired with each other in the “urls.py” python file. Each pairing is given a name so they may be easily referenced throughout the application. For example, in Figure 24, the URL “node/view/32” would match the first function, which calls the “node_views.view” controller.



```
# node URLs
url(r'^node/view/(?P<node_id>[0-9]+)/$', node_views.view, name='node_view'),
url(r'^node/edit/(?P<node_id>[0-9]+)/$', node_views.edit, name='node_edit'),
url(r'^node/create/$', node_views.create, name='node_create'),
url(r'^node/user/$', node_views.user, name='node_user'),
```

Figure 24. Example of URL-controller matching.

Remote Management

Since the sensor nodes are connected to IP end-to-end, we can remotely manage them through the web application. The web application sends commands through the control client that are received by the control server (see 3.4) on a sensor node. This interaction of components is described in Figure 25.

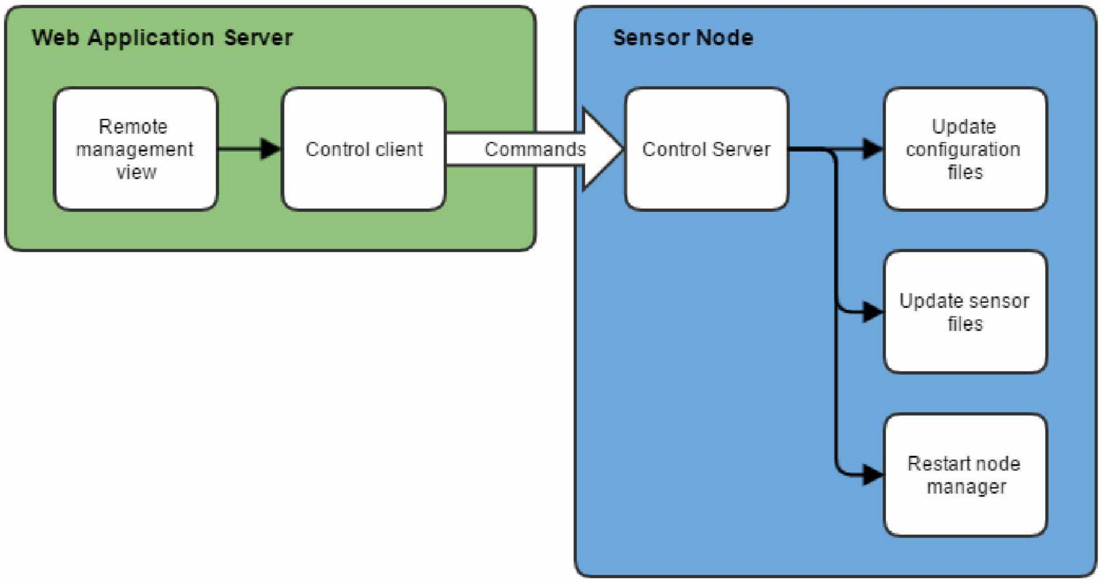


Figure 25. Remote management interactions.

The control client is called for the following operations:

Operation	Restart required?
A new sensor has been added to the sensor node.	Yes

A sensor package has been updated.	Yes
The node owner/node group member has changed a node configuration setting.	If requested.
The node owner/node group member requests a node manager restart.	Yes
The node owner/node group member creates an event.	Yes

Figure 26. Table describing the remote management operations.

When one of the following operations occurs on the web application, the control client will generate the JSON commands and send them to a single or multiple sensor nodes. If a connection error occurs with any of the sensor nodes, an alert will be placed on the node page and the node owner will be notified. The node owner may ask the control client to resend a command once the connection issue has been resolved.

Data Event Handling

The web application provides a simple event handling service for researchers. A user can create an Event, associating a Trigger with a server Action. This feature can help researchers discover misbehaving sensors or software errors faster by setting a trigger for data readings outside a reasonable range and an action for email notification.

Triggers are defined by the sensor author and input into the system after a sensor has been created and uploaded. From the sensor edit page, the sensor author can add triggers to be used by researchers. Actions are predefined routines to take place on the server. Currently sending an email is only supported.

Events can be associated with one or more nodes that are collecting data from the specified sensor. When the web application receives data from a sensor node, the Events are pulled and if any triggers are fired, the associated action will take place.

3.6 Data Analytics

CI Rainbow provides the user with the ability to perform data analytics on the data collected in the system using popular Python modules. We support data analytics through private Jupyter notebook instances, where the user may use the tools python provides in a web application interface.

Jupyter Notebook

Jupyter notebook is itself a web application for creating and sharing documents that contain: live code, equations, visualizations, and explanatory text [JUPYTER WEBSITE]. The Jupyter program starts a web application on a specified port and configuration file and a researcher can connect to it through their web browser. The installation is configured to support Python version 3.6 with several scientific modules: numpy, matplotlib, and scipy.

User Directory

In order to the store the various files generated by Jupyter notebook and any additional requirements for python modules, a separate directory is created for each individual user. For security reasons, this directory should be treated as the root directory for that user in order to protect the files on the system.

Web Application Interactions

The web application manages four interactions for the data analytics tools:

- Jupyter notebook instances.
- Importing data to Jupyter.
- Saving and loading the user's work.

The process for starting up a Jupyter notebook instance is shown in Figure 27. When the user requests a Jupyter instance, the web application checks to see if the authenticated user has a directory in the system already. If not, a new directory is created. The directory is used for storing Jupyter notebook information and any other files required for the user. Data can be imported through a MySQL user with read-only privileges listed on the web application's Jupyter launch page.

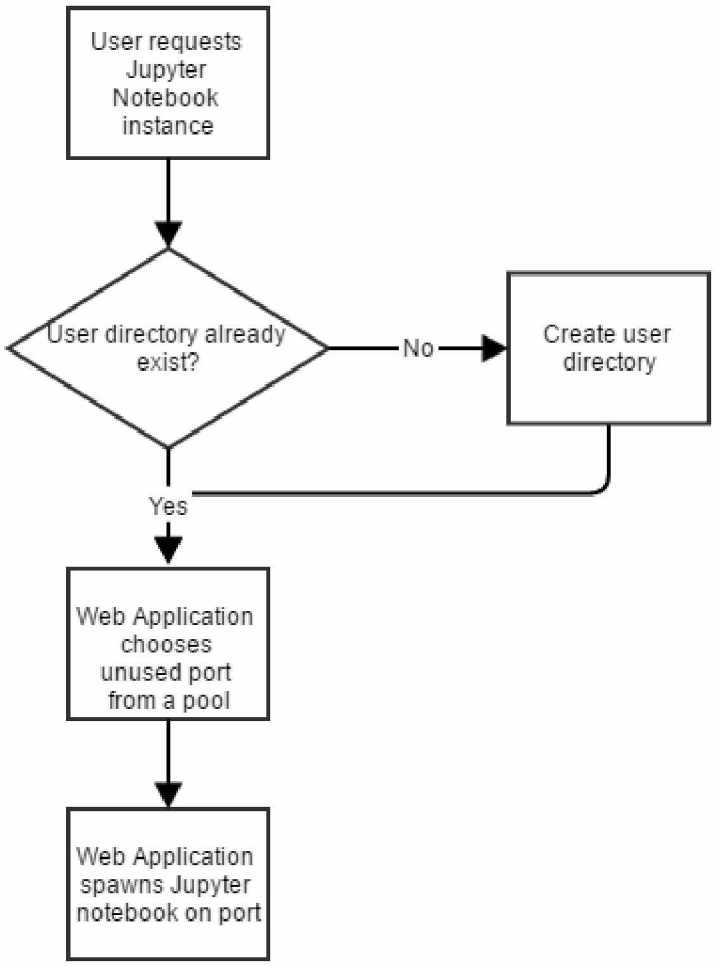

User requests Jupyter Notebook instance
User directory already exist?
No
Create user directory
Yes
Web Application chooses unused port from a pool
Web Application spawns Jupyter notebook on port

Figure 27. Flowchart of Jupyter notebook startup process.

Chapter 4: Deployment Details

In this chapter we discuss the capabilities of the hardware and the results from the software components of the system deployed in CI Park.

4.1 Infrastructure

Network Layout

The table in Figure 28 shows the details of the Ubiquity infrastructure installed to CI Park. Each location, save for Ironwood Hall, contains two routers with static IP addresses.

DNS Name	IP Address	Location	Antenna Type
ap01	172.30.239.10	Ironwood Hall	Directional
ap02	172.30.239.12	Water tower, facing Ironwood Hall	Directional
ap03	172.30.239.13	Water tower, facing the Ridge	Directional
ap04	172.30.239.14	Ridge, facing the Water Tower	Directional
ap05	172.30.239.15	Ridge, facing West Park	Omnidirectional
ap06	172.30.239.16	West Park, facing the Ridge	Directional
ap07	172.30.239.17	West Park	WiMAX
ap08	172.30.239.18	Central Park, facing the Ridge	Directional
ap09	172.30.239.19	Central Park	WiMAX
ap10	172.30.239.20	East Park, facing the Ridge	Directional
ap11	172.30.239.21	East Park	WiMAX

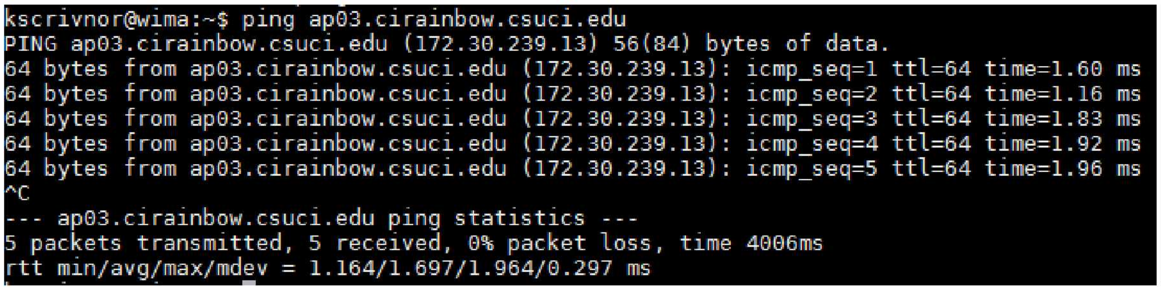
Figure 28. Table of Infrastructure details.

The 172.30.239.* internet is a private network that is only accessible through a direction connection in the lab or from the WIMA web application server (172.30.239.28). The WIMA web application server is connected to both the CI Rainbow private internet and the Internet.

Communication

Since each access point is accessible through *.cirainbow.csuci.edu, we can ping each point occasionally to test if it is currently accessible through the network. Figure 29 shows an example of pinging one of the access points from the WIMA server.

Any sensor node connected to the system is assigned an IP address via DHCP from one of the WiMAX routers in CI Park.



```
kscrivnor@wima:~$ ping ap03.cirainbow.csuci.edu
PING ap03.cirainbow.csuci.edu (172.30.239.13) 56(84) bytes of data.
64 bytes from ap03.cirainbow.csuci.edu (172.30.239.13): icmp_seq=1 ttl=64 time=1.60 ms
64 bytes from ap03.cirainbow.csuci.edu (172.30.239.13): icmp_seq=2 ttl=64 time=1.16 ms
64 bytes from ap03.cirainbow.csuci.edu (172.30.239.13): icmp_seq=3 ttl=64 time=1.83 ms
64 bytes from ap03.cirainbow.csuci.edu (172.30.239.13): icmp_seq=4 ttl=64 time=1.92 ms
64 bytes from ap03.cirainbow.csuci.edu (172.30.239.13): icmp_seq=5 ttl=64 time=1.96 ms
^C
--- ap03.cirainbow.csuci.edu ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 1.164/1.697/1.964/0.297 ms
```

Figure 29. Pinging the ridge facing water tower router.

4.2 Sensors

The system has been generically designed to handle data from most sensors. This includes both sensors with small textual data as well as sensors that require larger files, such as images or sound clips. Several undergraduate students have developed sensors to be used in CI Rainbow if needed. This includes: temperature, humidity, an anemometer, ground moisture, sound detector, and a camera sensor. For the initial deployment, we chose to use the temperature sensor as it was the simplest to waterproof. However, we will briefly discuss a few of the other sensors developed for the system.

The first two sensors, temperature and anemometer, collect data on a constant interval and the polling of the data is handled by the Node Manager. The second two, sound and camera sensors, were designed to collect data independently based on events. The Node Manager handled sending the data for all sensors, but was not required to poll the sound and camera sensor.

Temperature Sensor

The sensor used is a DS18B20 temperature sensor with waterproof casing. This sensor records data digitally in a two-byte register with increments down to 0.0625 Celsius. The 12-bit of temperature data provides a range of -55 to 125 Celsius. The power supply range is from 3.0V to 5.5V which the Raspberry Pi is capable of providing [21]. This sensor was used for the initial CI Park deployment.

Anemometer

An anemometer is a device for measuring wind speed. We chose to develop this sensor since Santa Rosa Island can become very windy. The sensor used is a generic anemometer sensor found on the adafruit website [22]. It is an analog sensor that returns a voltage value range of 0.4V to 2.0V representing 0.5 meters per second to 50 meters per second. The sensor does require additional voltage that the Raspberry Pi could not provide, so another battery was used to provide power for the anemometer.

Sound Sensor

A sound sensor was also developed with the motivation of capturing bird songs. The data collected by this sensor required use of the uploading media feature in CI Rainbow since the file sizes were potentially large. The sensor collected data on an interval but was also designed to collect data using feature vector extraction from yafee. A generic USB microphone sensor was used to collect the sound data.

Camera Sensor

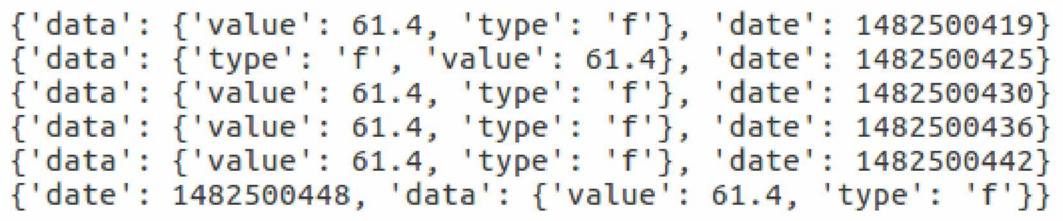
A camera sensor was developed to capture images of island foxes. If movement was detected by the sensor, an image would be placed in the media folder for uploading. The camera sensor was configurable through a configuration file for the following variables: cooldown time, PIR sensor polling interval, video or image mode, width and height dimensions for the images or videos, and how long to record a video for.

4.3 Sensor Node

The sensor software runs on a Raspberry Pi Zero. It is powered 3.7 Volt, 6600 mAh battery that is charged by a 6 Volt 6 Watt solar panel. A charger board is connected to the Raspberry Pi, battery, and solar panel where power is drawn automatically from the optimal source. The circuit components are contained in a weather proof case. For initial deployment, the DS18B20 temperature sensor was used.

4.4 Data Collection

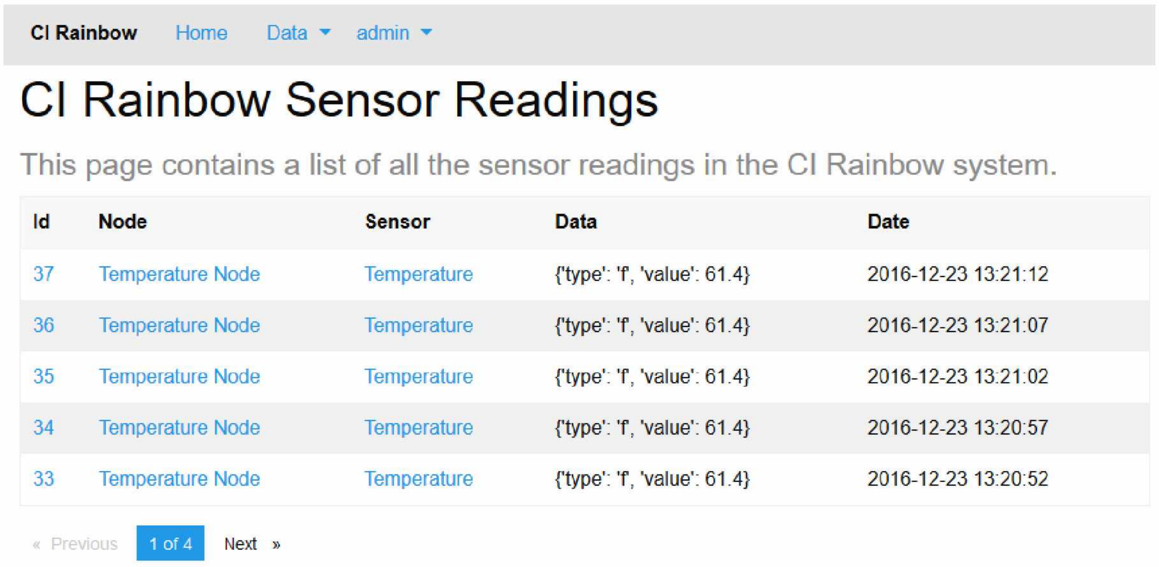
The temperature sensor node was tested and configured to poll the sensor every five seconds and upload the data every minute. A snippet of the data file for the sensor is shown in Figure 30. See Figure 12 for a description of how the data is organized. Note that Python treats JSON objects as dictionaries, since the keys are hashed, the ordering may vary.



```
{'data': {'value': 61.4, 'type': 'f'}, 'date': 1482500419}
{'data': {'type': 'f', 'value': 61.4}, 'date': 1482500425}
{'data': {'value': 61.4, 'type': 'f'}, 'date': 1482500430}
{'data': {'value': 61.4, 'type': 'f'}, 'date': 1482500436}
{'data': {'value': 61.4, 'type': 'f'}, 'date': 1482500442}
{'date': 1482500448, 'data': {'value': 61.4, 'type': 'f'}}
```

Figure 30. A snippet of temperature data from temperature.dat.

Once the Node Manager uploads the data to the Web Application, it can be viewed on the Reading view page shown in Figure 31.



CI Rainbow Sensor Readings

This page contains a list of all the sensor readings in the CI Rainbow system.

Id	Node	Sensor	Data	Date
37	Temperature Node	Temperature	{'type': 'f', 'value': 61.4}	2016-12-23 13:21:12
36	Temperature Node	Temperature	{'type': 'f', 'value': 61.4}	2016-12-23 13:21:07
35	Temperature Node	Temperature	{'type': 'f', 'value': 61.4}	2016-12-23 13:21:02
34	Temperature Node	Temperature	{'type': 'f', 'value': 61.4}	2016-12-23 13:20:57
33	Temperature Node	Temperature	{'type': 'f', 'value': 61.4}	2016-12-23 13:20:52

Figure 31. The Sensor Readings view from the Web Application.

Chapter 5: Analysis of CI Rainbow

In Chapter 4 we presented the deployment details of the CI Rainbow system. In this section we analyze the results with some additional perspective in relation to previously implemented WSNs.

5.1 Infrastructure

We decided to use a permanent infrastructure to connect the Sensor Nodes to the private CI Rainbow network. This makes long range communication between users/applications and the sensor nodes feasible. Since there is network connectivity throughout the entire system, we were able to implement features such as remote management, sensor updates, and near real time data collection. We have shown that it is possible to provide these features in a geological environment similar to that of Santa Rosa Island over a great distance.

There are some limitations imposed by using this type of infrastructure. Any area where data needs to be collected must have a line of sight to one of the long distance nodes. To expand the system to include a new section, new antenna must be installed to connect it to the network. However, the initial cost of adding to the infrastructure could potentially be made up by requiring less man hours to constantly retrieve sensor data, especially if the location is very remote and isolated.

5.2 Linux Based

Our sensor nodes are Linux based and that provides the opportunity for developers to use any language supported on the Linux OS to create a sensor. The sound sensor for instance, was written in C++ and used two additional software packages for the feature extraction and sound recording. The temperature, anemometer, and camera sensor have been written in Python 3. The node manager is not required to know which programming language a sensor has been implemented with and this design decision gives CI Rainbow the advantage of a more open platform for data collection. By emulating a sensor node's OS, compile times and development can be done at speeds far greater than provide by the Raspberry Pi hardware. Disk images can be created of this environment and then run on the Raspberry Pi after development and testing.

While using Linux as the operating system for our sensor nodes provides many useful features and conveniences for developers, there are some drawbacks to this approach. Running a full-fledged distribution of Linux requires additional computation and therefore power consumption that we have not measured the effect of. Linux requires processes to run that may not be useful for simply collecting data. Many WSNs use TinyOS, contiki or another WSN specific operating system to run their systems. These have the advantage of being designed specifically for use in a WSN environment.

5.3 Generic Sensors

The CI Rainbow system requires all data to be collected and presented in a standard format. By providing this standard format for collecting data, all data in the system may be treated equally among researchers. This gives the advantage for researchers who want to perform data mining techniques mixing data from different researcher's projects.

All of the WSN we have presented in Chapter 2 are designed specifically for collecting data for one or more type of sensors. Our goal was to create a system that can be expanded to collect data with many sensors with the only requirement being to write code for a new sensor. With the CI Rainbow system already in place, additional data can be collected without the need of creating a new WSN specific for that sensor.

5.4 Remote Management

With the persistent connectivity from the infrastructure we can provide remote management features to the system. It is possible to update sensor code if there are any bugs discovered after deployment. The standardization of the configuration files is another advantage of using the system. Since all configuration files are created with the same format, we can remotely modify any sensor configuration file in the system through the control server. We believe this is a useful feature for researchers who made need to change sensor parameters on the fly and we have not seen this in any reviewed literature.

5.5 Near Real Time Data Collection

Near real time results can be obtained through the system if configured to send data at short intervals. Since the network infrastructure directly connects sensor nodes to the web application server, researchers can obtain data results from sensors in near real time. Once a sensor node communicates the data to the web application, it is immediately available for viewing. The ability to provide this feature is a direct consequence of the permanent network infrastructure supporting the system. Other WSN, such as The Great Duck Island (GDI) WSN [15] described in Chapter 2, have also provided this feature.

5.6 Data Format

The data collected is presented in its raw JSON format for any sensor. It is a requirement for sensor authors to save the data in the format specified in Chapter 3. While every WSN has a unique method for storing data specific to its sensors, we have provided a standard format for data collected from any sensor in the system.

Chapter 6: Conclusion

The main goals of this research and development was to create a WSN system with three main ideas: non sensor specific design, sensor node management, and a Linux development environment.

Most WSNs developed have little to no infrastructure and lack the persistent connection that our system has. By using a permanent network infrastructure, this it has opened the possibilities for remote management not generally seen in a typical WSN. The ability to update sensor files and configurations during testing greatly aided the testing efforts for sensor development. We believe that this feature will be useful for researchers in the future.

With the recent developments of the Raspberry Pi's capabilities and size, running a Linux distribution as the operating system for a sensor node is not so out of reach as it used to be. Even if the power requirements may be greater, the conveniences provided by Linux allow us to create a flexible system.

We have shown that it is possible for the system to work with many different types of sensors developed independently by undergraduate students. Currently we can deploy temperature, ground moisture, image and video capture, sound, anemometer, and island fox RFID sensors to the system. Each of these sensors collects data in the JSON format described earlier.

By demonstrating a working system, it is our hope that we can deploy CI Rainbow to the Santa Rosa Island research station and begin collecting data for researchers.

Chapter 7: Future Work

Ultimately, we want to deploy CI Rainbow to serve the California State University Channel Islands Santa Rosa Research Station for researchers in Biology and ESRM to collect data remotely. The purpose of the CI Park exploratory network was to test and evaluate the system and identify any additional work or fixes that need to be completed.

7.1 Power Consumption

Infrastructure

The network infrastructure maintained connection and acceptable uptime throughout the initial testing and deployment. However, we have not collected any data on the total power consumption for each of the components throughout the system. Some additional work is required to state whether the solar and battery setup will maintain enough power to support the system fully with minimal downtime.

Sensor Node

Sensor Node power consumption will also vary depending on the sensors attached, polling time, and other variables. While the sensor node with a temperate sensor maintained uptime throughout our experiment window, we do not have data on the power consumption during this period. Additional testing of how many power is required will be needed before a deployment to the Santa Rosa island.

There are power saving options available on the Raspberry Pi Zero hardware that will offer an additional advantage in power consumption. It is possible to disable the HDMI output and LEDs to save on energy not required in a deployment.

7.2 Remote Management Additions

Currently the remote management of nodes is limited in its capabilities. Simple configuration updates and sensor additions are supported, however, since the systems are running Linux much more complex abilities can be added for remote management. These additions should include the ability to update the system packages, add/remove files, perform system upgrades, and possibly update system configuration files.

7.3 Enhanced Data Views

All data is treated equally on the system so the data is always displayed in its raw form. For some data, it would be convenient to display the data with some additional visuals such as heat maps, graphs, or histograms. Since the scope of this thesis was to obtain data remotely, we chose to simply present it in a simple HTML table. For researchers adding sensors to the system in the future, the ability to customize the data views will prove useful.

7.4 Security

Following common practices for a secure web applications have been followed throughout the development process of CI Rainbow. However, the focus of the work was to provide a proof of concept system showing the abilities in a closed university environment. In this section we consider some of the potential security risks that came up during design and testing.

Anyone can sign up and add nodes/sensors to the system. Obviously this is not ideal and a potential solution is to disable sign ups and create an invite only system. There are limitations as to what can be accomplished by a user not associated with any existing groups in place. So a malicious user cannot harm other users work directly other than creating a node and uploading fake data for another user's sensor.

The REST API calls require a secure token to be attached to each transaction, however all transactions currently occur through the HTTP. A full deployment should incorporate HTTPS into all REST API calls to ensure that the data passed through is encrypted.

The CI Rainbow network is a private internet that is only accessible through two data ports on CSUCI. However, one of these is the web application server that bridges the gap between the Internet and the CI Rainbow network. If the web application is compromised in any way, the rest of the network should be considered to be compromised.

7.5 Sensor Node Hardware

The system has been designed to be flexible with sensors as long as the sensor author follows the conventions of the sensor node software. Since the web application was also designed with this idea, it would be possible for hardware other than the Raspberry Pi to be added for the CI Rainbow system. We have not explored this option since it is beyond the scope of this thesis.

7.6 Node Actions

Currently the system supports server Actions from Events that run server side, such as sending emails or text messages. However, there are use cases where an action should take place on the sensor node itself. For instance, if an RFID sensor detects a fox is near, this should trigger the camera sensor to turn on until the RFID sensor loses contact. This was not implemented fully due to time constraints.

7.7 Time Synchronization

The time is managed by the Linux operating system but it is possible for a researcher to require more stringent time synchronization across the system. Since it is a built in feature in the operating system we have not included any additional work for time synchronization, however many other WSNs have [23] [24].

References

[1] P. Zhang, C. M. Sadler, A. S. Lyon and M. Martonosi, "Hardware Design Experiences in ZebraNet," *SenSys,* pp. 227-238, 2004.

[2] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin, "A Wireless Sensor Network For Structural Monitoring," *SenSys,* pp. 13-24, 2004.

[3] K. Martinez, R. Ong and J. Hart, "Glacsweb: A Sensor Network for Hostile Environments," *IEEE Xplore,* 2004.

[4] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young and J. Porter, "LUSTER: Wireless Sensor Network for Environmental Researc," *SenSys,* 2007.

[5] J. Yick, B. Mukherjee and D. Ghosal, "Wireless Sensor Network Survey," *Computer Networks 52,* pp. 2292-2330, 2008.

[6] P. Rawat, K. D. Singh, H. Chaouchi and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *The Journal of Supercomputing,* pp. 1-48, 2014.

[7] H. P and R. Das, Wireless Sensor Networks (WSN) 2014-2024: Forecasts, Technologies, Players, IDTechEx, 2015.

[8] A. A. Abbasi and M. Younis, "A Survey on Clustering Algorithms for Wireless Sensor Networks," *Computer Communications,* vol. 30, no. 14-15, pp. 2826-2841, 2007.

[9] D. Goyal and M. R. Tripathy, "Routing Protocols in Wireless Sensor Networks: A Survey," in *Advanced Computing & Communication Technologies,* 2012.

[10] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks," *IPDPS,* vol. 1, p. 186, 2001.

[11] A. Perrig, J. Stankovic and D. Wagner, "Security in Wireless Sensor Networks," *Communcations of the ACM,* vol. 47, no. 6, pp. 53-57, 2004.

[12] M. Vieira, C. N. Coelho, D. C. da Silva and J. M. de Mata, "Survey on Wireless Sensor Network Devices," in *Emerging Technologies and Factory Automation*, 2003.

[13] S.-H. Yang, Wireless Sensor Networks: Principles, Design and Applications, London: Springer London, 2014.

[14] M. Delamo, S. Felici-Castell, J. J. Perez-Solano and A. Foster, "Designing an open source maintenance-free Environmental Monitoring Application for Wireless Sensor Networks," *The Journal of Systems and Software,* vol. 103, pp. 238-247, 2015.

[15] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *WSNA '02 Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, New York, 2002.

[16] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay and W. Hong, "A Macroscope in the Redwoods," in *SenSys '05 Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, 2005.

[17] V. Gallart, S. Felici-Castell, M. Delamo, A. Foster and J. J. Perez, "Evaluation of a Real, Low Cost, Urban WSN Deployment for Accurate Environmental Monitoring," *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on,* pp. 634-639, 2011.

[18] J. K. Hart and K. Martinez, "Environmental Sensor Networks: A revolution in the earth system science?," *Earth-Science Reviews,* vol. 78, no. 3-4, pp. 177-191, 2006.

[19] L. M. L. Oliveira and J. J. P. C. Rodrigues, "Wireless Sensor Networks: a Survey on Environmental Monitoring," *Journal of Communications,* vol. 6, no. 2, pp. 143-151, 2011.

[20] R. T. Fielding, Architectural styles and the design of network-based software architectures, irvine: University of California, Irvine, 2000.

[21] *DS18B20 Programmable Resolution 1-Wire Digital Thermometer,* Sunnyvale: Maxim Integrated Products, 2008.

[22] "Anemometer Wind Speed Sensor w/Analog Voltage Output," [Online]. Available: https://www.adafruit.com/products/1733. [Accessed 1 12 2016].

[23] S. Ganeriwal, R. Kumar and M. Srivastava, "Time-sync Protocol for Sensor Networks," *SenSys '03,* pp. 138-149, 2003.

[24] J. Elson, L. Girod and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," in *5th Symposium on Operating Systems Design and Implementation*, New York, 2002.