engineering proceedings

MDPI

Proceeding Paper

A Distributed Sensor Network (DSN) Employing a Raspberry Pi 3 Model B Microprocessor and a Custom-Designed and Factory-Manufactured Multi-Purpose Printed Circuit Board for Future Sensing Projects †

Alan Ibbett and Yeslam Al-Saggaf *

School of Computing, Mathematics and Engineering, Charles Sturt University, Wagga Wagga, NSW 2678, Australia; aibbett@csu.edu.au

* Correspondence: yalsaggaf@csu.edu.au

† Presented at the 10th International Electronic Conference on Sensors and Applications (ECSA-10), 15–30 November 2023; Available online: https://ecsa-10.sciforum.net/.

Abstract: This paper presents a detailed design of an inexpensive, simple, and scalable Distributed Sensor Network (DSN). Each sensor's hardware consists of a Raspberry Pi 3 Model B microprocessor, a specifically designed and factory-made Printed Circuit Board (PCB), an Uninterruptible Power Supply (UPS) Hat based on a High-Capacity Lithium Polymer battery (LiPo), a Power over Ethernet Splitter, a GPS receiver, and a LoRaWAN module. Each sensor is built to capture GPS, Wi-Fi, and Bluetooth signals and sends this information to a network controller implementing a LoRaWAN gateway. Each sensor's software is developed so that all applications run on top of a Linux operating system. The layer above it includes system daemon applications, such as Air-mon, HCI tools, GPSd, and networking support. An SQLite database sits on top of the daemon applications and records the captured information. After the DSN was successfully tested, it was deployed in a research study. The novelty of this study is that this was the first time that a DSN was used in high schools to detect leakage from IoT devices to educate students about cyber safety.

Keywords: distributed sensor network; Raspberry Pi; printed circuit board; LoRaWAN gateway; inter-integrated circuit

Citation: Ibbett, A.; Al-Saggaf, Y. A Distributed Sensor Network (DSN) Employing a Raspberry Pi 3 Model B Microprocessor and a Custom-Designed and Factory-Manufactured Multi-Purpose Printed Circuit Board for Future Sensing Projects. *Eng. Proc.* **2023**, *58*, 55. https://doi.org/10.3390/ecsa-10-16187

Academic Editor: Stefano Mariani

Published: 15 November 2023

1. Introduction

This paper presents the detailed design of the various components of a Distributed Sensor Network (DSN). The DSN design is inspired by the WiFiScanMap [1]. The paper begins by first describing the design of the sensor nodes and then the network controller. The paper then presents the key features of the software design of the nodes and the software for the dashboards. The paper then reports the results of a novel study in which the DSN was used in a school setting to educate students about cyber safety.

The DSN comprises a number of nodes, designed to monitor Wi-Fi and Bluetooth traffic within the range of the node [2]. Each node is an independent module, and each node controls and manages information flows between the node and the DSN network controller [2]. Control and management information shared between nodes and the DSN controller includes node status and other telemetry information [2]. Figure 1 below depicts the overall architecture of the DSN.

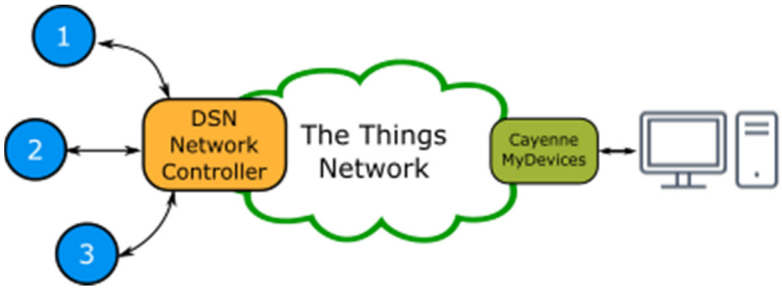

Figure 1. Key design blocks of the DSN [2].

The sensor network monitors Wi-Fi and Bluetooth connection requests, beacons and other broadcast-type packets, including the Basic Service Set Identifier (BSSID), Service Set Identifier (SSID), Media Access Control address of client (MAC), Channel Number, Received Signal Strength Information (RSSI), Encryption Capabilities, UTC time and date of capture, and location information of capture [2].

To capture and manage these packets, a number of open-source tools were used, including iwlist (an application to display detailed wireless information), airmon-ng (a packet capturing applications for Linux), gpsd gps (GPS support for Linux), SQLite (an open-source database), python 3.0 (an open-source programming language), and DB Browser for SQLite (an open-source database browser). Other supporting software such as the Re4son Kernel and the Raspbian Linux distribution were also used to support the DSN [2].

2. Hardware Design

Each sensor's hardware consists of a Raspberry Pi 3 Model B microprocessor, a specifically designed and factory-made Printed Circuit Board (PCB), an Uninterruptible Power Supply (UPS) Hat based on a High-Capacity Lithium Polymer battery (LiPo), a Power over Ethernet Splitter allowing for 5 V power via micro-USB (from a PoE injector), a GPS receiver, a LoRaWAN module, control push buttons, an OLED screen, a Four-wire I2C connector, a 40-pin Raspberry Pi connector, a status LED, and five GPIO connectors configured for Signal, Voltage, and Ground (SVG) [2]. Each sensor is built to capture GPS, Wi-Fi, and Bluetooth signals and sends this information to a network controller implementing a LoRaWAN gateway [2].

The Raspberry Pi Model 3B microprocessor had Wi-Fi and Bluetooth built onto the main board. The Wi-Fi standard is based on the frequency of operation being in the so-called Industrial Scientific and Medical (ISM) band, which is unlicensed and free to use. The ISM band covers many frequencies, and Wi-Fi has been defined to work on the 2.4 GHz and 5 GHz part of the ISM bands. The Wi-Fi on the Raspberry Pi 3B was limited to the 2.4 Ghz band [2].

A prototype Printed Circuit Board (PCB) was designed and sent for printing to OSH-PARK, a PCB fabrication house based in the United States that specializes in producing high-quality low-cost small-batch runs of PCBs. All node components were designed to be on a single PCB that can be stacked onto other boards as required [2]. By using this "stackable" architecture, it was possible to quickly assemble and, if necessary, disassemble a node. The sensor board [2] contains the following items:

- Organic light-emitting diode (OLED) display to indicate system status: the OLED provides a way to display a simple menu-based user interface;
- RFM9x LoRa radio: this provides a network connection to LoRaWAN or other LoRa nodes;
- Global Position System GPS receiver: this allows the node to accurately collect coordinated universal time (UTC) time and date as well as location information of other data captured from the GPS;
- DC power supply input connector to supply regulated 5VDC to the system;

- Status LED to show system status at a glance;
- Five GPIO ports of the Raspberry Pi have been broken up into an SVG layout for connecting to external devices. These can include relays, sensors, switches, and so forth;
- A three-button keypad for interacting with the OLED menu user interface.

The addition of I2C and GPIO breakout connections meant the node could be used with a wide variety of sensors in a number of remote locations [2].

The DSN node has been designed to be self-contained and to be used in a fixed location [2]. Uninterruptable power supplies (UPSs) based on the high-capacity 3.7 Lithium Polymer (LiPo) battery technology provided power supply to the central processing unit (CPU) and sensors. LiPo batteries are high-capacity, easy to use, and relatively inexpensive. The LiPo battery is paired with a purpose-built UPS card (or "hat", as expansion cards for the Raspberry Pi are known) that has the appropriate circuitry to detect a low battery level and initiate a controlled shutdown of the node. The UPS card also handles the charging of the unit. Five volts from the PoE splitter are plugged into the GPS unit which then feeds the entire node [2]. Figure 2 shows the key elements of the DSN node before assembly. Figure 3 shows the elements of the DSN node after assembly.

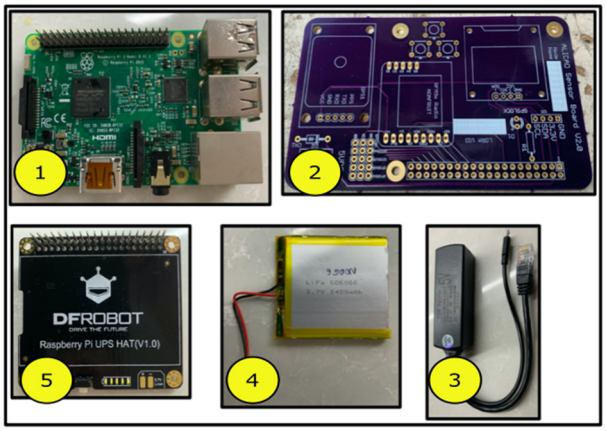

Figure 2. The key elements of the DSN node. Taken from top row moving clockwise: (1) CPU (Raspberry Pi Model 3B); (2) Sensor PCB, shown here unpopulated; (3) Power over Ethernet Splitter; (4) High-Capacity LiPo battery; (5) UPS Hat [2].

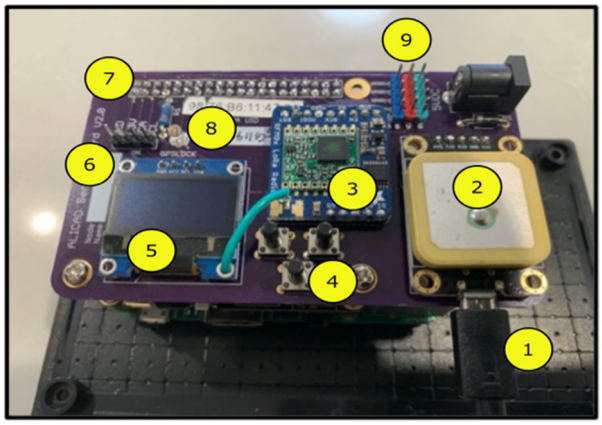

Figure 3. A complete DSN node. (1) 5V power in via micro-USB from the PoE injector; (2) GPS unit (the chip antenna is visible in the picture); (3) LoRaWAN module; (4) Control push buttons; (5) OLED screen; (6) Four-wire I2C connector; (7) 40-pin Raspberry Pi connector; (8) Status LED; (9) Five GPIO connectors configured for Signal, Voltage, and Ground (SVG) [2].

As the nodes in this design were placed in fixed locations, an "out of band" monitoring service was needed to check the status of each node. In this case, LoRa radios and the LoRaWAN network were chosen to provide this remote monitoring [2]. The LoRa network is completely separate in frequency and operation to both the Wi-Fi and Bluetooth networks that the nodes are monitoring, thereby separating node management functions from the networks being monitored [2].

The DSN network controller (DSNNC) is designed to provide real-time feedback on the operation of each of the distributed sensor nodes. The role of the DSNNC is to act as a gateway from each of the DSN nodes to a central dashboard. Here, the status of each node can be monitored and, if necessary, the node can be restarted [2]. LoRa is a simple technology and permits easy integration of telemetry and control data. LoRa is also completely out of band, which means that LoRa transmissions do not share the same IP network space as the Wi-Fi that the DSN is monitoring [3].

The RAK7249 LoRaWAN gateway provided a wide range of connectivity and power supply options including Gigabit Ethernet, Wi-Fi, 4G, and PoE power supply [2]. The gateway's geographical and time values are set by an integrated GPS receiver. The gateway has a (LoRa) range of over 15 km. The free Cayenne MyDevices (CMD) dashboard service was selected to provide the application server services. The CMD was selected because it is secure, easy to set up, allows for shared viewing of dashboards, and is free [2].

3. Software Design

The original Lauters code [1] published on GitHub was used as the template for the software design after translating the Python 2.0 code to Python 3.0, but it was necessary to add in additional features to support the DSN hardware, including (1) LoRa support, (2) Display Support, (3) Menu Support, and (4) Keypad Support [2]. As the nodes were placed in fixed locations, in order to allow each node to start up autonomously, it was necessary to create a start-up script that was unique to each node [2].

The software architecture of a DSN node is simple (see Figure 4 below). All applications run on top of the Linux operating system [2]. For additional information, see the Supplementary Materials section below. The next layer includes system daemon applications such as Air-mon, HCI tools, GPSd, and networking support. The SQLite database sits on top of the daemon applications, and information is delivered to the user by the web server. Networking tools are made available to all layers of the system. The web server is implemented by a custom Python script that also controls the way that the daemon processes access the database [2].

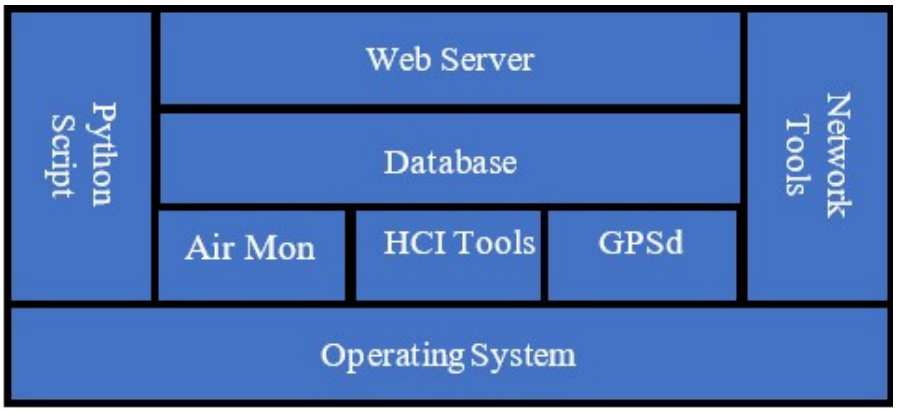

Figure 4. DSN node software architecture [2].

The nodes are designed to capture and log GPS, Wi-Fi, and Bluetooth transmission, but provision has been made to connect other digital instruments for monitoring and logging via an Inter-Integrated Circuit (I2C) and one-wire data busses [2].

The DSN node is a general-purpose device and can support a number of different software options [2]. The DSN node is based on a Linux Distribution with a number of local tasks running such as Virtual Network Computing (VNC), as well as the main data acquisition task. The main data acquisition task is written as a single class in Python 3.0. The instantiation of this class provides all of the data structure initialization required, as well as the database creation and setup if needed [2].

There are five tasks associated with the DSN implementation, namely, the GPS Poller, Wi-Fi Poller, Bluetooth Poller, LCD Poller, and LoRa Poller [2], as depicted in Figure 5. The GPS Poller task runs to regularly collect, parse, and validate GPS data. The validated data are passed back to the main thread where they are written to the database. The GPS Poller thread looks for latitude and longitudinal data, as well as precision and time information. The Wi-Fi Poller thread uses data collected from the airmon-ng process to collect Wi-Fi packets from the surrounding area. These packets are validated by the thread and passed back to the main thread. The Bluetooth Poller thread is similar to the Wi-Fi Poller thread but uses the hcitool to capture Bluetooth packets [2].

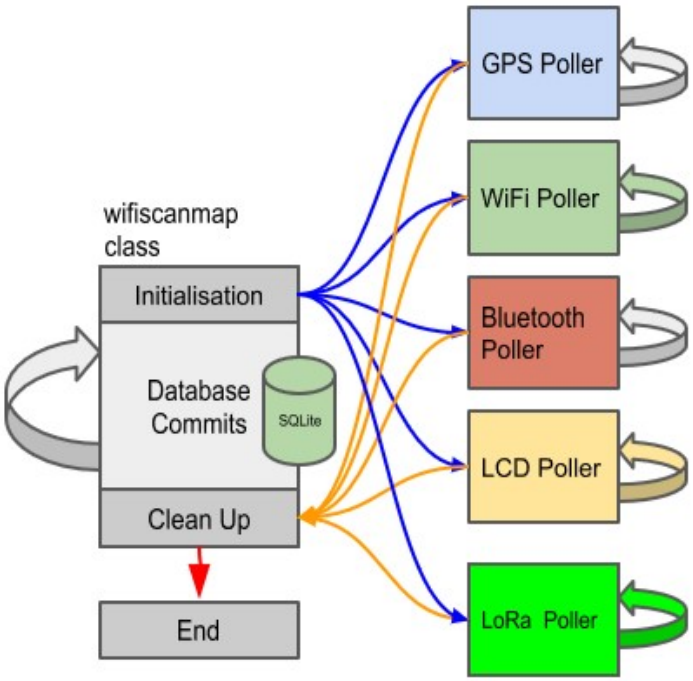

Figure 5. DSN node software architecture [2].

The LCD Poller task handles the LCD and the user interface push buttons [2]. The LCD Poller thread uses the rpi-128 × 64-oled -menu Python library [4]. This library, in turn, uses the RPi.GPIO library. A number of simple menus were set up to allow the user to check the state of the DSN node. The reliability of the data gathered by this experiment depends on the ability of the DSN to capture and record packets. The status of this packet capture was monitored by the DSNNC. The DSNNC regularly sends LoRa messages to the user via the RAK gateway. If these signals are lost, or indicate that a node has failed, then two actions can occur: the node will attempt to reconnect with the controller and, if necessary, will reboot [2].

The Wi-Fi and Bluetooth Poller threads rely on the Airmon-ng script [2]. Airmon-ng enables monitor mode on suitable wireless interfaces. The script also allows an interface to be switched from monitor mode (the mode used in this application) to managed mode (traditional Wi-Fi mode). Airmon-ng allows for the capture and classification of packets. Airmon-ng requires that the Wi-Fi chip on the Raspberry Pi be put into monitoring mode, which was achieved with the help of the drivers from the nexmon project [5]. More details on the re4son project can be found here [6]. This workaround allowed the Airmon-ng code

to collect Wi-Fi packets as expected. These data were then recorded into an SQLite Database, which sat on top of the daemon applications. Networking tools are made available to all layers of the system. Information from each sensor forming the DSN that is received by the LoRaWAN gateway is communicated to the user via a web server. The web server is implemented by a custom Python script which also controlled the way that the daemon processes accessed the database [2].

4. Study Results and Conclusions

A series of tests were carried out to ensure that the DSN was functioning as expected [2]. Based on the successful outcome of these tests, the DSN was used in a first research study of its kind, in which the DSN was deployed on a high school ground in Australia to educate students about cyber safety [2]. After informing the students about the presence of a DSN on campus detecting leakage from IoT devices, the DSN was turned on. The traffic that was captured by the DSN at that time served as the standard for a normal traffic at that school. Then, during a cyber safety lesson, the students were shown the results of the DSN monitoring and were trained on how to modify their device settings to reduce data leakage. At that time, the DSN continued to detect leakage from IoT devices to see if children changed their device settings to protect themselves online. The results of the study revealed that the amount of data leaked from smartphones after the cyber safety lesson was significantly smaller than the traffic captured before the cyber safety lesson, suggesting that the cyber safety lesson was effective in reducing data leakage [2].

The DSN was simple, scalable, and inexpensive. The total cost to build a single node (a sensor), including the cost of the Raspberry Pi, was less than AUD 160 [2]. The inclusion of a LoRaWAN module, control push buttons, OLED screen, and status LED made accessing and controlling the DSN remotely a simple process [2]. The scalability of the sensor's PCB, which has a Four-wire I2C connector, makes the device capable of performing numerous types of sensing and control functions.

Supplementary Materials: The following supporting information can be downloaded at https://github.com/alanibbett (accessed on 8 October 2023). Files S1: Distributed Sensor Network—PCB Files; Files S2: Distributed Sensor Network Python Script.

Author Contributions: Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, and visualization A.I.; writing—original draft preparation, writing—review and editing, and supervision Y.A.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: This study was conducted in accordance with the Declaration of Helsinki and approved by the Human Research Ethics Committee) of Charles Sturt University (protocol code H19072 and date of approval 6 June 2020).

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article and supplementary materials.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bordeaux: A Digital Urban Exploration. Available online: https://github.com/mehdilauters/wifiScanMap/blob/master/Results.md (accessed on 8 October 2023).
2. Ibbett, A. An Examination of Real-World Data Leakage from IoT Devices. Ph.D. Thesis, Charles Sturt University, Wagga Wagga, Australia, 2022.
3. Vangelista, L.; Zanella, A.; Zorzi, M. Long-Range IoT Technologies: The Dawn of LoRa™. In *Future Access Enablers for Ubiquitous and Intelligent Infrastructures: Proceedings of the First International Conference, FABULOUS 2015, Ohrid, Republic of Macedonia, 23–25 September 2015*; Revised Selected Papers 2015; Springer International Publishing: Berlin/Heidelberg, Germany, 2015; pp. 51–58. [CrossRef]
4. rpi-128x64-Oled-Menusystem. Available online: https://github.com/jpuk/rpi-128x64-oled-menusystem (accessed on 8 October 2023).

5. Schulz, M.; Wegemer, D.; Hollick, M. Nexmon: Build your own wi-fi testbeds with low-level mac and phy-access using firmware patches on off-the-shelf mobile devices. In Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, Snowbird, UT, USA, 20 October 2017.
6. Boeving, C.; Klimaszewski, S.; Rebillout, A.; Wilson, B.; Ruiz de Alegría, D.; O'Gorman, J.; O'Gorman, J.; Hertzog, R.; Re4son-Kernel for Raspberry Pi. Kali Linux Development Team. 12 May 2019. Available online: https://re4son-kernel.com/re4son-pi-kernel/ (accessed on 8 October 2023).