

Edinburgh Napier
University

Honours Project.

ENG10100

**Carbon Dioxide and Oxygen
Internet of Things (IoT)
Monitoring System.**

Olivier Chaligne (B.Eng Mechatronics) - 40292302
Dr Chan Hwang See - Supervisor

Abstract:

This report offers an overview of the work undertaken to develop an Internet of Thing application - a wireless sensor network able to measure carbon dioxide and oxygen within an environment.

Project aims and objectives:

- Review of the carbon cycle with impact on the environment and health.
- Overview of the Internet of Things solutions and relevant standards.
- Technical description of the development and operation of the wireless sensors networks for the purpose of measuring carbon dioxide and oxygen.
- Cloud management/display of collected data.

Project Conclusions:

- Carbon dioxide and air pollution in general, have an enormous negative impact on nearly every organ in the human body.
- Using the sinking capability of nature can help mitigate the effects of carbon dioxide and pollutants whilst restoring soil quality.
- The topic of IoT is easier to break down in terms of: Communication Technologies, Standards and Software/Network protocol stacks rather than by Consortiums and Alliances.
- This report gives a detailed account of the implementation and operation of the MRF24J40MA Radio Frequency Transceiver (IEEE 802.15.4 compatible) which would be a good guide for someone developing a similar application or using this module.

Table Of Content

Design Brief - Problem Statement.	5
Project Background Research: Carbon and Internet of Things.	6
Carbon Dioxide, Public & Environmental Health and Carbon Capture & Storage.	6
Air Pollution and Public Health.	6
Carbon: Land Degradation and Desertification.	8
Carbon Capture and Storage/Sinking.	9
Network of Things (Internet of Things) and IoT Solutions Providers.	13
Internet and Network of Things and Wireless Sensors Networks.	13
Market Segmentation: Consortiums and Alliances.	14
Wireless Communication Technologies.	15
IEEE 802.15.4 Standard: MQTT, MiWi and ZigBee Network Protocols.	18
IEEE 802.15.4 Standard.	18
MQTT For Sensor Networks (MQTT-SN).	21
Microchip MiWi.	22
ZigBee.	23
Product Design Specifications.	24
Concept Development and Prototyping.	25
Components Description.	25
Arduino Microcontroller Platforms.	25
RF Transceiver MRF24J40MA.	26
Carbon Dioxide Sensor: CoZIR-AH-5000.	35
Oxygen Sensor: LuminOx Optical Oxygen.	37
Bi-Directional Logic Level Converter.	39
Power Supply and Sleep Mode Considerations.	39
Components Interfacing Methods.	44
Universal Asynchronous Receiver/Transmitter (UART).	44
Serial Peripheral Interface (SPI).	46
Application Software.	48
Libraries and Read/Write from/to Control Registers.	48
Setup and Initialisation phase.	52
Obtaining Sensor Data.	54
Packaging and Sending.	54
Receiving and Reading Data.	55
Activities undertaken during prototyping.	56
CoZIR Sensor Basic Measurement.	56
Integrating the MRF24J40MA Transceiver.	59
Arduino MKR WiFi 1010 Connecting to the Cloud.	65
Results and Prototype Evaluation.	68
References.	70
Appendices.	77

List of Symbols and Abbreviations:

CCA - Clear Channel Assessment.

CCS - Carbon Capture and Storage.

CO₂ - Carbon Dioxide.

CRC - Cyclic Redundancy Check.

CS - Chip Select.

CSMA-CA - Carrier Sense Multiple Access/Collision Avoidance.

ED - Energy Detection.

FCS - Frame Check Sequence.

FFD - Full Function Device.

FIFO - First In First Out.

INT - Interrupt.

IoT - Internet of Things.

Kbps - kilobits per second.

LPWA - Low-Power Wide-Area.

LQI - Link Quality Indicator.

LR - Low-Rate.

LSB - Least Significant Bit.

mA - milliamp.

MAC - Medium Access Control.

mAh - milliamp hour.

mbar - millibar.

Mbps - Megabit per second.

MCU - Microcontroller Unit.

MISO - Master In Slave Out.

MOSI - Master Out Slave In.

MQTT SN - Message Queuing Telemetry Transport - Sensor Network

MSB - Most Significant Bit.

NB IoT - Narrowband IoT.

NDIR - Non Dispersive Infrared.

NoT - Network of Things.

O₂ - Oxygen.

PHR - PHY Header.

PHY - physical interface.

PLL - Phase-Locked Loop.

ppm - parts per million.

RF - Radio Frequency.

RFD - Reduced Function Device.

RFI - Radio Frequency Interference.
 RH% - Relative Humidity.
 RST - Reset.
 RSSI - Received Signal Strength Indicator.
 SCLK - Serial Clock.
 SFD - Start-of-Frame Delimiter.
 SHR - Synchronisation Header
 SS - Slave Select.
 SPI - Serial Peripheral Interface.
 UART - Universal Asynchronous Receiver/Transmitter.
 VCO - Voltage-Controlled Oscillator.
 WPAN - Wireless Personal Area Network.
 WSN - Wireless Sensors Network.

List of Tables and Illustrations:

Fig I.A.3.a Acorn a Low-cost, Low-risk Catalyst For Clean Growth	10
Fig I.A.3.b Major carbon pools and fluxes of the global carbon balance - Carbon sequestration in dryland soils.	11
Fig I.A.3.c Carbon balance within the soil (brown box) is controlled by carbon inputs from photosynthesis and carbon losses by respiration.	12
Fig I.B.3.a Two Leading LPWA Technologies	16
Fig I.B.3.b LTE-M Key Applications	17
Fig I.B.3.c NB-IoT Key Points	17
Fig I.C.1.a MAC layer frame structure	19
Fig I.C.1.b Physical layer packet structure.	19
Fig I.C.1.c Superframes	20
Fig I.C.1.d Star and peer-to-peer topology examples	20
Fig I.C.1.e Cluster tree network	21
Table V.A.2.a Channel Selection RFCON0 (0x200) register setting	30
Fig. V.A.2.b: TX Normal FIFO Format	31
Fig V.A.2.c RSSI vs. RECEIVED POWER (dBm)	34
Table V.A.6.a Sensor Node Main Electronic Components Power Consumption	41
Table V.A.6.b Gateway Coordinator Main Electronic Components Power Consumption	43
Fig V.B.a Serial Communication Port	44
Fig V.B.1.a RS-232 Serial Packet Basic Structure.	45
Fig V.B.1.b Packet Structure for “9” and “10”	45
Figure V.B.2.a Serial Peripheral Interface with 3 Slave Devices	46
Table V.B.2.b The Different SPI Clock Modes	47

Table V.D.1.b Arduino and CoZIR-AH-5000 Wiring	56
Table V.D.1.b CoZIR-AH-5000 Pin Layout	56
Fig V.D.1.b CoZIR-AH-5000 connector pins	57
Fig V.D.1.c CozIR-AH-5000 Connected to Arduino Uno	57
Fig V.D.1.d Serial Monitor CO2 Reading from CozIR sensor Output	58
Fig V.D.2.a MRF24J40MA and Arduino Nano Every Pin Header Assembly	59
Table V.D.2.a Arduino-MRF24J40MA Pin Connections	60
Fig V.D.2.b Sensor Node (Arduino Nano Every) Breadboard Prototype	61
Fig V.D.2c Gateway Coordinator/Relay (Arduino Nano Every) Breadboard Prototype	62
Fig V.D.2.d Gateway Coordinator (Arduino MKR WiFi 1010) Breadboard Prototype	62
Fig V.D.2.e Voltage Level Arduino-MRF24J40MA (without bi-directional logic level converter)	63
Fig V.D.2.f Voltage Level Arduino-MRF24J40MA (with bi-directional logic level converter)	63
Fig V.D.2.e COM16 Sensor Node Serial Monitor/COM14 Gateway Serial Monitor	64
Fig.V.D.3.a MKR ENV Shield and Arduino MKR WiFi 1010	65
Fig V.D.3.b Network Scanning - Console Output	66
Fig V.D.3.c Serial Monitor - Sensor Readings.	66
Fig V.D.3.d Serial Monitor - Connecting to Arduino Cloud	67
Fig V.D.3.e Arduino IoT Cloud Dashboard - Sensor Readings Display.	67

Acknowledgements.

I would like to thank Dr.See, my supervisor, for the guidance and the encouragements I received throughout the project.

I would like to also thank the Student Grant Initiative Fund for providing funding which enabled the procurement of some of the equipment and components used to develop the application.

In 1956, The UK Parliament and Government put in place the Clean Air Act as a response to the London Great Smog of 1952 and its repercussions on the population. Since then, the Act has been reviewed until 1993 (date of the last update). Although the Smog phenomena is an event from the past in the UK (still occurs in countries such as China and India), calls have been made to review and update the Clean Air Act (Davis, 2017). The aim is to make fit for the health challenges (Carrington et al., 2018) but also to tackle the effects of climate change posed by air pollution (Soil erosion, desertification, ocean acidification, global warming...)

The purpose of this study is to explore the development of an Internet of Things application which could help to understand and mitigate the effects of air pollution. This report provides an account of the tasks undertaken to develop the prototype of a wireless sensors network to monitor carbon dioxide and oxygen within the environment.

Although the term “Internet of Things” first appeared in 1999 (Foote, 2016) with the wireless sensors network concept going back to the 1950s for military purposes (Silicon Labs, 2013), applications based on these technologies are only starting to make their way in various fields (Agriculture, Health, Manufacturing...) very recently. Affordable Internet of Things solutions have appeared in the last few years which make them cost efficient tools. It is known from research that carbon dioxide is the main gas contributing to climate change with a contribution estimated at 64% (European Environment Agency, 2016). It is released in the environment in many ways (Fossil fuels consumption and cutting trees for example). Observing its behaviour in various environments may provide a solution to reduce public exposure to the gas but also to identify environments propitious to trapping the carbon.

Project aims and objectives:

- Review of the carbon cycle with impact on the environment and health.
- Overview of the Internet of Things solutions and relevant standards.
- Technical description of the development and operation of the wireless sensors networks for the purpose of measuring carbon dioxide and oxygen.
- Cloud management/display of collected data.

I. Design Brief - Problem Statement.

The design brief given below offers a general overview of the application to be performed by the prototype. It serves as a general basis to the work conducted presented in this report.

The prototype should provide a network of wireless sensors including sensor nodes, relays and gateways. Each element of the network should function off the grid and be low powered to offer the longest lifetime without maintenance. The purpose of the applications is to provide measurements for carbon dioxide and oxygen levels within an environment (indoor and outdoor). The measurements should be returned and made available online in a database for further processing.

II. Project Background Research: Carbon and Internet of Things.

A. Carbon Dioxide, Public & Environmental Health and Carbon Capture & Storage.

This section will provide context to the domain of application of the prototype developed through a review of carbon dioxide, its implication on the environment & health and solutions to mitigate its effects is given.

1. Air Pollution and Public Health.

In January 2020, the British Heart Foundation launched their campaign “You’re Full of it” aimed at raising awareness amongst the public about the impact of air pollution and the levels of toxic particles stored as a consequence in the human body. The research they founded predicts an average of 160,000 people will die by 2030 due to exposure to air pollution. (BHF, 2020)

The World Health Organisation has called air pollution the new tobacco because of its indiscriminate effect on the population and more noticeably on children. It also warned that 91% of the world population lived in areas with above WHO limits in terms of air pollution. (Carrington and Taylor, 2018)

Carrington et al (2018) reported on research led by the Forum of International Respiratory Societies’ Environmental Committee (Schraufnagel, Dean E. et al, 2019).

The article highlights some of the impact on health of air pollution:

- Lungs and Heart: Asthma to emphysema, lung cancer and heart attacks...
- Brain and Mind: Strokes, dementia, reduced intelligence, depression, poor sleep...
- Abdominal organs: Cancers and IBS.
- Cellular, Bone and Skin Structure.
- Reproduction and Children: Fertility, miscarriages, stunted growth, leukaemia, mental health problems...

Despite targets and strategies available, it is reported most urban areas in the UK still have illegal levels of air pollution (Carrington and Taylor, 2018).

The UK's Clean Air Strategy over the years has been praised by the World Health Organisation as an example for the world to follow (PA Media, 2020). This strategy has been pushed by national institutions such as: The UK Government, The MET Office and UK Research & Innovation (UKRI).

Through its Clean Air Programme UKRI aims to support health and clean growth through inter and multidisciplinary international research focused on developing practical applications to tackling future air quality related issues.

Through the financing available from Strategic Priorities Fund the programme is split in two waves:

First wave (£20.5m): Focused on near-term outdoor solutions to develop short-term policies, manage non-exhaust transport emissions and pilot a framework for clean air analysis.

Second wave (£22m): Aimed at tackling emerging air issues relating to new pollutants (indoor and outdoor) and the impact caused by their exposure, with a focus on vulnerable groups.

The specific aims of the second wave are:

- Build an interdisciplinary focused on indoor/outdoor air quality emission, exposure and health impacts.
- Uncover new knowledge about emerging air pollution and health risks to limit exposure through the use of communication and technology.
- Provide evidence based recommendations to develop regulations and policies.
- Stimulate business innovation in the sustainability products and services sectors aiming at protecting health.

The funding available is also used to support coordinators who facilitate joint-working and identify areas of common interest to rationalise and maximise research efforts.

(UKRI, 2019)

2. Carbon: Land Degradation and Desertification.

The MET Office predicts that in 2020 the atmospheric concentration of carbon dioxide will rise by 10% (MET, 2019). Although the rise is mainly due to fossil fuel combustion and cement production (67%), land degradation also has a big impact (33%) on the levels of carbon dioxide being released in the atmosphere from the soil representing “annual global emissions of up to 4.4 billion tonnes of CO₂ between 2000 and 2009” (Priestley International Centre for Climate, 2018). It is recognised soil with a higher carbon content increases the organic matter and quality of the ground (Food And Agriculture Organization Of The United Nations, 2004). This demonstrates how the unbalance of carbon dioxide between the atmosphere and soil negatively impacts the environment, climate change and human activity.

Land degradation can be explained by human activities which pollute or degrade the soil quality such as: Industrial agriculture, cities development, deforestation, high levels of consumption (in developed and rising in developing economies) . Soil damage can also be triggered by natural causes, for example droughts and coastal surges which salinate land.(WHO, 2012)

The World Health Organisation (2012), the Priestley International Centre for Climate (2018) and Food And Agriculture Organization Of The United Nations (2020) define the risks and effects of the land degradation as:

- Reduced amount of food (reduced crop yields and healthy soil farming land) and water.
- Diseases because of the lack of clean water.
- Respiratory issues because of dust due to erosion and pollutants in the atmosphere.
- Spread of diseases due to migration.
- Reduction/destruction of biodiversity and wildlife.

3. Carbon Capture and Storage/Sinking.

Many options are being deployed against carbon dioxide and other greenhouse gases to remove them from the atmosphere or mitigate their effects. The Carbon Capture and Storage Association (2011) offers a definition for industrially engineered technologies falling under the term Carbon Capture and Storage (CCS) also called Atmospheric Carbon Capture (ACCA) (Institute for Materials and Processes, n.d.): "...is a technology that can capture up to 90% of the carbon dioxide (CO₂) emissions produced from the use of fossil fuels in electricity generation and industrial processes, preventing the carbon dioxide from entering the atmosphere."

The purpose is to extract carbon dioxide from the atmosphere and then store it. Projects like ACT Acorn (2018) aim to make storage available, using existing oil & gas infrastructure, offshore in the North East of Scotland in disused reservoirs (as shown in Fig I.A.3.a below)

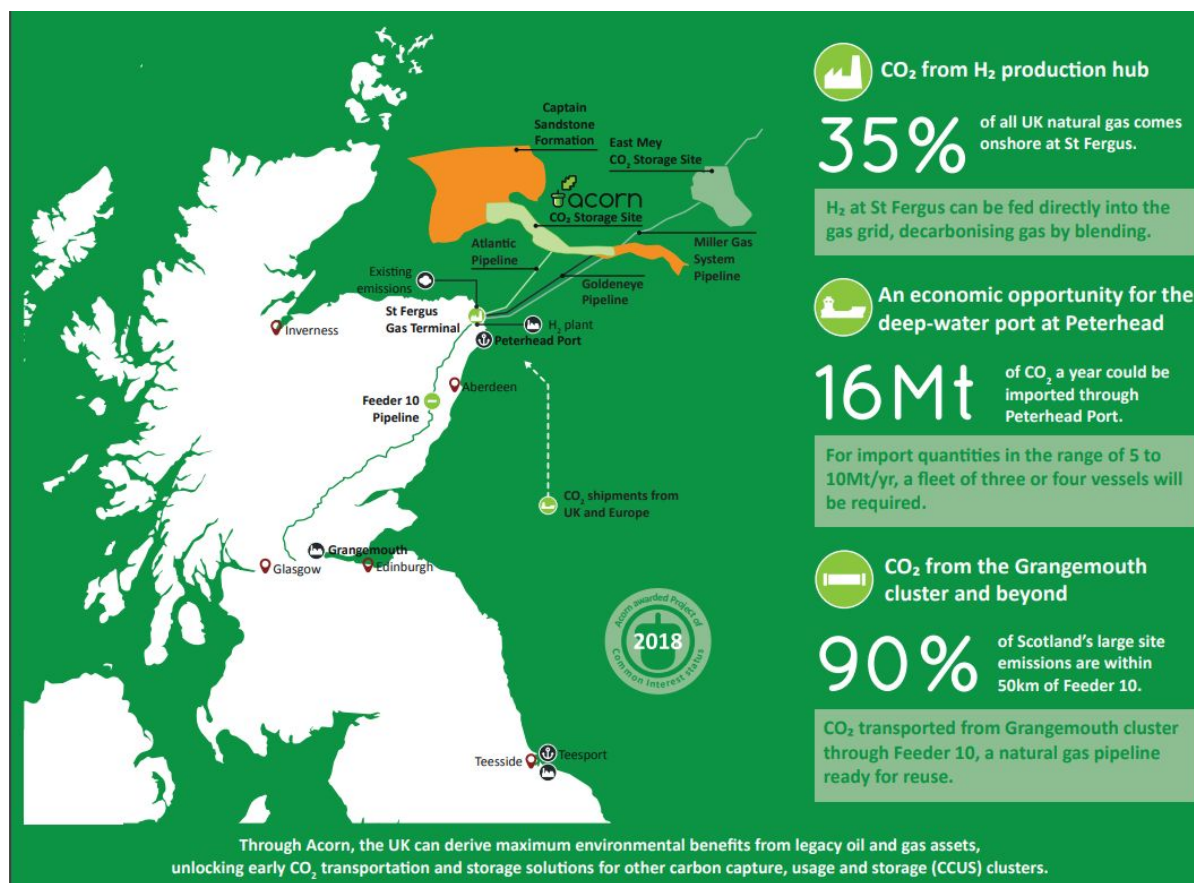


Fig I.A.3.a Acorn a Low-cost, Low-risk Catalyst For Clean Growth. (SCCS, 2019)

The downside of CCS is the important amount of energy used to capture and store the carbon dioxide despite the efficacy of the process. Another option is to use natural carbon sinks. The European Environment Agency (2020) defines carbon sinks as: “Forests and other ecosystems that absorb carbon, thereby removing it from the atmosphere and offsetting CO₂ emissions.” (European Environment Agency, 2020). Sinks are natural systems found in nature which exchange CO₂ with the atmosphere as shown in Fig I.A.3.b

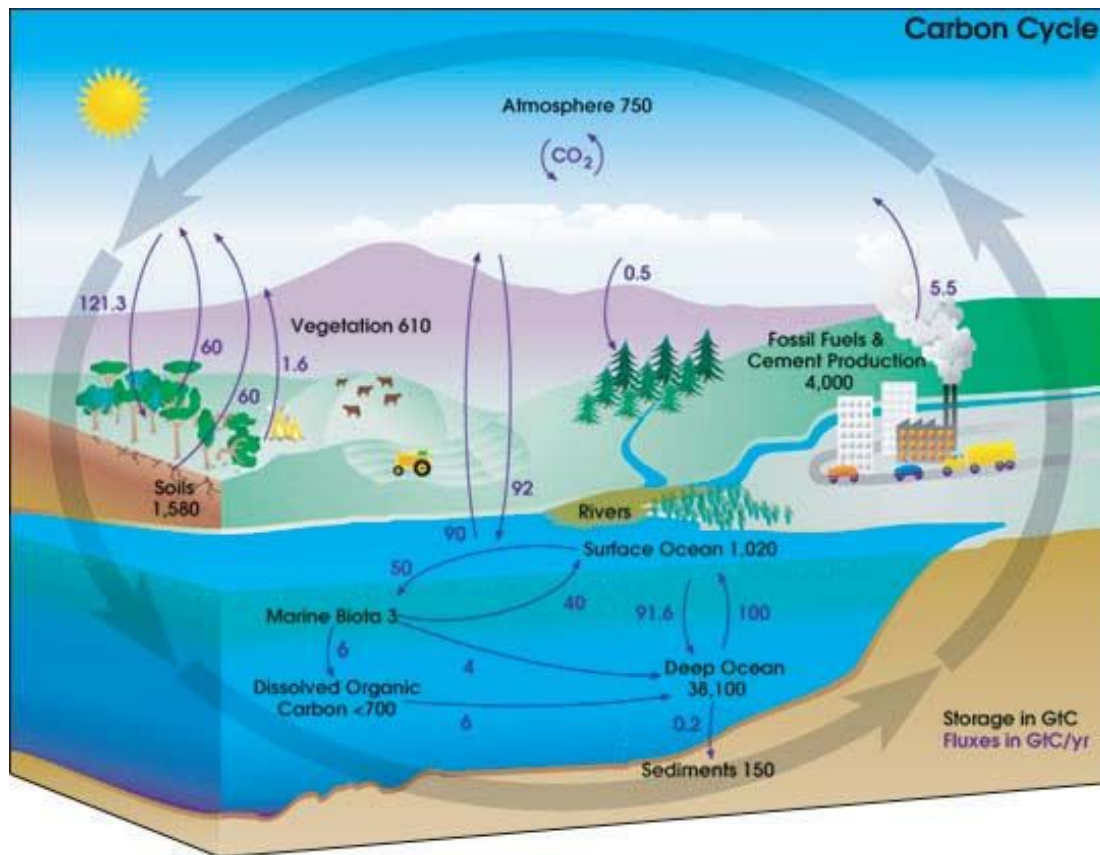


Fig I.A.3.b Major carbon pools and fluxes of the global carbon balance - Carbon sequestration in dryland soils. (Food And Agriculture Organization Of The United Nations, 2020)

Franzluëbbers and Doraiswamy (2007), the Food And Agriculture Organization Of The United Nations (2020) and Ontl and Schulte (2012) demonstrate that by adopting responsible agricultural practices and programmes aimed protecting and developing natural carbon sinks, and the process of earth respiration (as described in Fig I.A.3.c - next page), land degradation can be reversed and more carbon dioxide can be captured in a beneficial way.

The International Union for Conservation of Nature - IUCN (2015), believes that an increase of the overall carbon storage by 1% in the top meter of soil would represent more than the annual emission CO₂ due to fossil fuel burning.

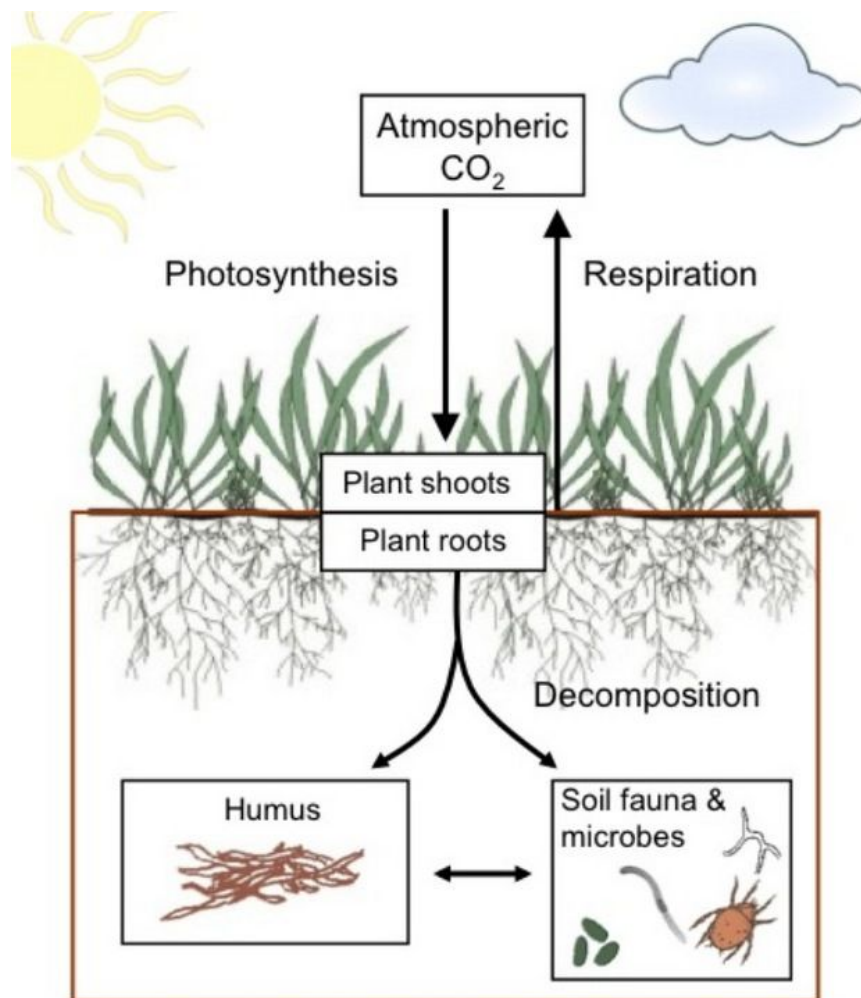


Fig I.A.3.c Carbon balance within the soil (brown box) is controlled by carbon inputs from photosynthesis and carbon losses by respiration. (Decomposition of roots and root products by soil fauna and microbes produces humus, a long-lived store of Soil Organic Carbon).

(Ontl and Schulte, 2012)

B. Network of Things (Internet of Things) and IoT Solutions Providers.

This section offers an overview of the Internet of Things: terminology, solutions and protocols (with an emphasis on options available for the development of the application presented in this report).

1. Internet and Network of Things and Wireless Sensors Networks.

Internet of Things to Network of Things - The domain of the Internet of Things (IoT) being relatively recent its definition is updated regularly. Jeffrey Voas (2016) jointly with NIST (2016) offers to rebrand IoT as “Network of Things” (NoT). The Internet is a network or channel but not all IoT (or NoT) applications utilise this network. Effectively NoT is a more appropriate term.

According to Voas and the NIST, NoTs are composed of these common characteristics:

- Sensors - measure physical property.
- Aggregators - Software “packaging” the data from raw to intermediate aggregated data.
- Communication Channel - Channel through which the data is carried.
- External Utilities - software or hardware product or service (databases, mobile devices, miscellaneous software or hardware systems, clouds, computers, CPUs...).
- Decision Triggers - conditional expression that triggers an action.

(J. Voas, B. Agresti and Laplante, 2018)

Wireless Sensors Network - In the domain of wireless sensor networks several IoT protocols have been applied to demonstrate the feasibility of such applications:

- Dr. See (2011) proves how a network of sensors using a ZigBee Mesh network can be used to monitor water-level in gully pots in an urban environment.

- Allahham and Rahman (2018) demonstrated how a ZigBee network was used to control doors on a campus.
- Butgereit and Nickless (2013) show how a network of sensors can be implemented to monitor CO2 movements off the coast of South Africa and transmit the data through Twitter.
- Dhingra, Madda, Gandomi, Patan and Daneshmand (2019) explain how using Arduinos and wifi modules, they were able to monitor air quality within a city and offer predictive journeys to minimise exposure to pollution through a mobile phone app.
- Gupta and Quan (2018) demonstrate how using several wireless sensors, it is possible to monitor environmental factors in a greenhouse.

2. Market Segmentation: Consortiums and Alliances.

There are many NoT/IoT solutions providers available on the market. These providers are often regrouped under Alliances and Consortiums. They can be classified between three categories: Technology development focused, focused on a specific domain of application/industry or education focused. They aim for their solution to become the standard to needs faced by the industry.

Examples of Alliances and Consortiums:

- IEEE: RFID Consortium, NFC Forum, Dash 7 Alliance, Bluetooth SIG, Wifi Alliance, Zigbee Alliance, The ULE Alliance, Wi-SUN Alliance, World Wide Web Consortium.
- The LoRa Alliance Wide Area networks: Actility, Bouygues Telecom, Cisco, FastNet, Eolane, IBM, Kerlink, KPN, IMST, Microchip Technology, MultiTech, Proximus, Sagemcom, Semtech, SingTel and Swisscom.
- The Weightless SIG: Founders - ARM, Cable & Wireless, Landis+Gyr and Neul.
- The Internet Engineering Task Force (IETF).
- ISO (International Organization for Standardization).

(Trsridhar, 2018)

3. Wireless Communication Technologies.

Wireless communication is a convenient way of transmitting information over the air this, simplifying implementation of technical solutions. This section will present some of the wireless technologies available.

WiFi - This is the most commonly recognised technology because of its implementation in private homes. It was developed by the Institute of Electrical and Electronics Engineer. It first appeared in 1997 under the IEEE 802.11 standard. Subsequent versions of the standard were created from 1999 to 2008: IEEE 802.11 (a.b.g.n.y).

WiFi operates on ISM Bands: 2.4 GHz, 3.7GHz and 5GHz with data rates of 2Mbps up to 248Mbps in a range of 20 to 5000 meters according to operating standard.

WiFi employs a star network topology meaning only the WiFi hub and network nodes can dialogue with each other (no internode communication).

(Wang, Jiang and Zhang, 2019)

Bluetooth - This another well known communication technology appearing in everyday applications to replace cables between fixed and portable devices (for example keyboards, headphones, midi instruments...). Bluetooth (IEEE 802.15.1) is a good example of a standard rapidly accepted and integrated to applications worldwide. It offers omni-directional transfer of data and sound simultaneously.

It operates in the 2.4GHz frequency band with data rates of: 723.2 kbps (one way) - 57.6kbps (return) or 433.9 kbps (symmetrically) in range of 1, 10 or 100 meters according to device class (1 to 3 respectively).

Bluetooth can operate Ad Hoc (Master-Slave), Piconet (same as star) or Scatternet (similar to mesh) networks.

(Wang, Jiang and Zhang, 2019)

Low-Rate (LR) WPAN (Wireless Personal Area Network) - This technology was introduced by the IEEE 802.15 standards and acts as a basis for several networking protocols (more on this in section I.C.1). It focuses on providing low cost and low power consumption transmission methods. The trade off is in the data rates which go from 20 kbps to 250 kbps. It operates in the 2.4GHz ISM, 915MHz (USA) and 868MHz (Europe) bands. The networks can be configured as: Star, Peer to Peer or Cluster Tree topologies.

(Wang, Jiang and Zhang, 2019)

Cellular IoT - This is the technology primarily employed in smartphones such as 2G, 3G and 4G. This allows it to reap the benefits of an already well established long range network infrastructure. Cellular IoT aims to provide a solution for the high power consumption of traditional cellular and offer an option for low or infrequent data transmission. (Rand, 2020)

Cellular IoT can be split between two main technologies LTE-M (Long Term Evolution - Machine) and NB-IoT (Narrowband IoT) as shown in Fig I.B.3.a below.

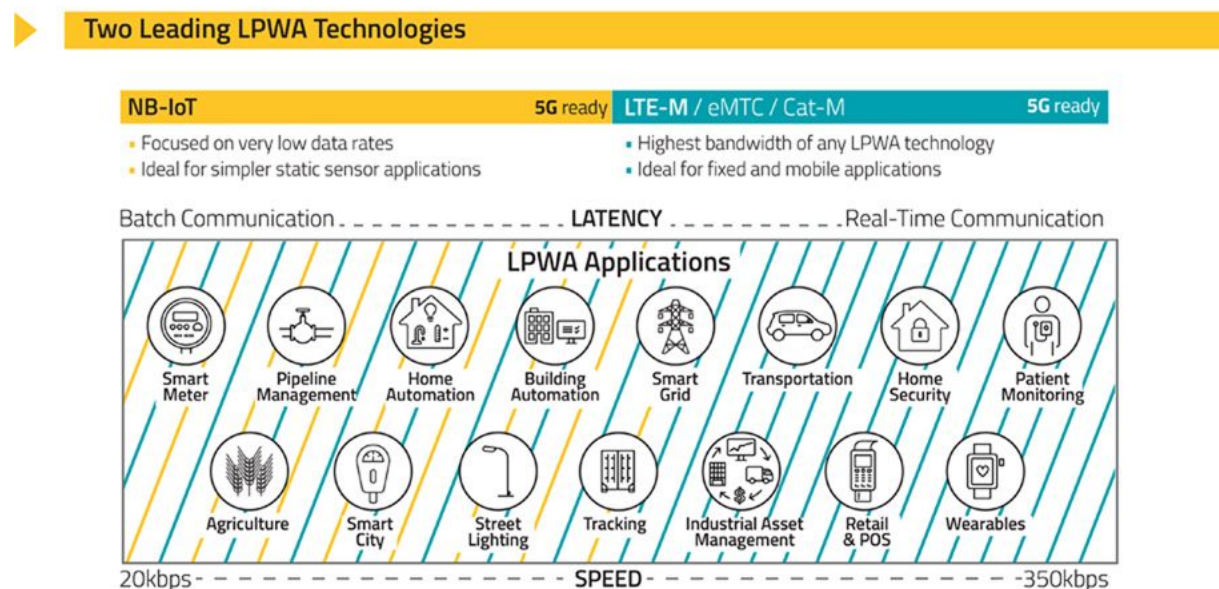


Fig I.B.3.a Two Leading LPWA Technologies (Hwang, 2020).

LTE-M: is a cellular technology appropriate for IoT and Machine to Machine (M2M) applications. It applies to low power, wide area networks for the transmission of a medium amount of data (as shown in Fig I.B.3.b) . It incorporates LTE Cat M1 (data rate both ways: 375 kbps) and its evolution LTE Cat M2 (data rates: download - 2.4 Mbps and upload - 2.6 Mbps). LTE M technology is especially useful for applications which require mobility because of its capacity of switching between cellular towers without having to reconnect.(u-blox, 2018a)

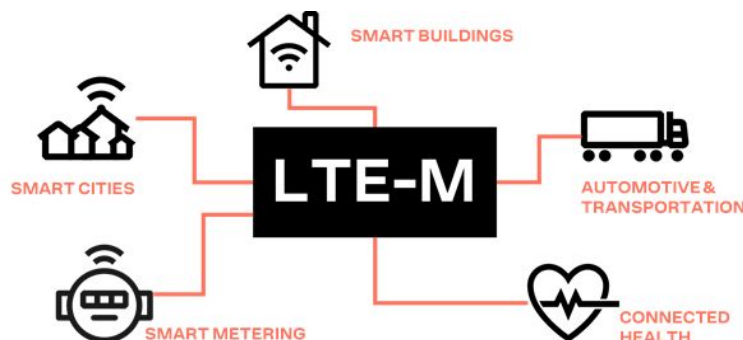


Fig I.B.3.b LTE-M Key Applications (u-blox, 2018a).

NB-IoT: Narrowband IoT is a low power wide area technology. Its main focus is to provide communication for applications requiring a low amount of data to be transferred (data rate: downloads - 125 kbps and uploads - 140 kbps). It is especially useful in areas where there are no cellular towers close by or in shielded environments. Moreover it offers power saving optimisations which are not available on other cellular technologies. (u-blox, 2018b)

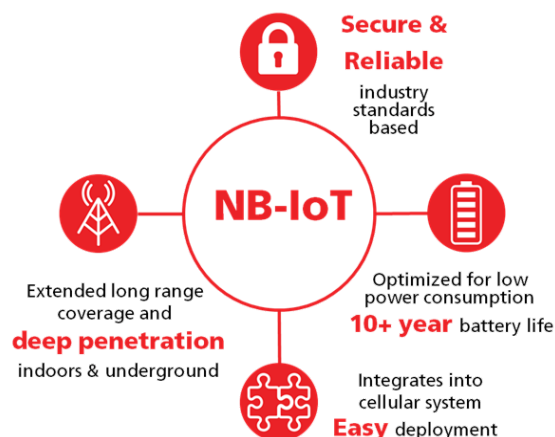


Fig I.B.3.c NB-IoT Key Points (u-blox, 2015).

C. IEEE 802.15.4 Standard: MQTT, MiWi and ZigBee Network Protocols.

This section describes the basic principles encapsulated by the IEEE 802.15.4 for wireless specifications for low-rate wireless personal area networks as introduced in section I.B.3 Low-Rate WPAN. Then, an overview of some compatible networking protocols acting as the top layer of the application will be given.

1. IEEE 802.15.4 Standard.

This standard first appeared in 2003 and was developed by a group at the IEEE. It describes the principles behind Low-Rate WPAN at two physical layers: 868/915 MHz (10 Channels) and 2.4GHz (16 Channels). Those bands respectively provide speeds up to 20 kbps, 40 kbps and 250 kbps. (Institute Of Electrical And Electronics Engineers, 2003; Devadiga, 2011)

In 802.15.4 standard, two types of devices are identified: full function (FFD) and Reduced function devices (RFD).

- FFDs can: be used in any type of networks, be PAN Coordinators, communicate with any other devices and can implement the whole protocol set.
- RFDs only can only implement a limited amount of the protocol set. Those are essentially end-devices in a star or peer-to-peer network.

The standard also identifies three devices types according to their function within the network:

- Network Device: FFD or RFD with adequate MAC and PHY information compatible with 802.15.4 specifications.
- Coordinator: FFD configured to synchronise applications within the network.
- PAN Coordinator: FFD- coordinator configured to be the main controller of the network.

(Institute Of Electrical And Electronics Engineers, 2003; Mirzoev, 2014)

As previously mentioned, devices require Medium Access Control (MAC) and physical interface (PHY) information to be recognised and operate within a network. Those information are stored in frames as shown in Fig I.C.1.a and I.C.1.b. The MAC layer is responsible for handling physical access to the radio channel and maintaining safe communication between devices. The PHY layer manages: data transmission and reception, energy detection (ED), link quality indication (LQI) and clear channel assessment (CCA).

Frame Control	Sequence Number	Destination Address	Source Address	Payload	Frame Check Sequence
2 bytes	1 byte	0-20 bytes		variable	2 bytes

Fig I.C.1.a MAC layer frame structure.

Preamble	Delimiter	Header	Physical Data Service Unit (PDSU)
4 bytes	1 byte	1 byte	≤ 127 bytes

Fig I.C.1.b Physical layer packet structure.

(Institute Of Electrical And Electronics Engineers, 2003; Devadiga, 2011)

To handle communication the devices utilise the Carrier Sensing Multiple Access with Collision Avoidance (CSMA-CA) algorithm. In 802.15.4, networks can be controlled by beacons or not.

In the case of a non-beacon enabled network, the CSMA-CA uses the unslotted mode. This mode waits a random amount of time before performing a Clear Channel Assessment (CCA). If the CCA returns and idle state the data will be transmitted. If not the process will be repeated.

If the network is beacon enabled, the CSMA-CA uses the slotted mechanism. The PAN Coordinator creates a superframe (Fig I.C.1.c) by dividing the time between two beacons into 16 time slots. The beacons contain the information about how the device can use the time slots to transfer data.

(Devadiga, 2011)

Beacon	Contention Access Period	Content Free Period			Inactive Period	Beacon
		Guaranteed Time Slot 1	Guaranteed Time Slot 2	GTS n...		

Fig I.C.1.c Superframes (Devadiga, 2011).

The 802.15.4 Standard offers three levels of security:

- No security
- Access control lists (ACL): Each device can contain 255 ACLs to control communication with the device.
- Advanced encryption standard (AES): providing confidentiality and/or integrity checks.

(Institute Of Electrical And Electronics Engineers, 2003; Devadiga, 2011)

Topologies for 802.15.4 Wireless Networks: Essentially there are two main types as shown in Fig I.C.1.d. Those can be combined to create more complex networks such as demonstrated in Fig I.C.1.e.

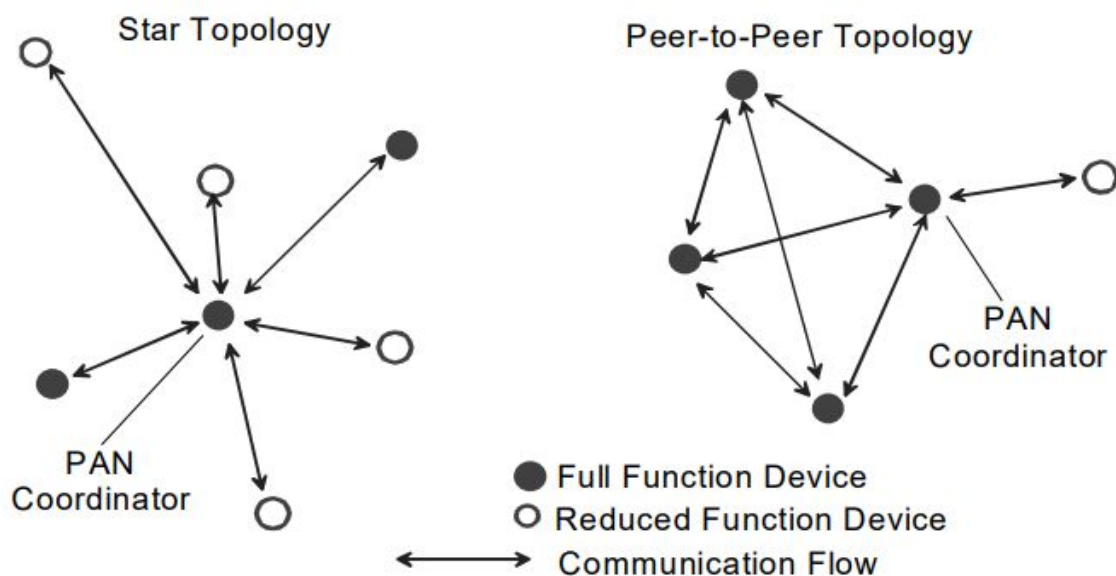


Fig I.C.1.d Star and peer-to-peer topology examples (Institute Of Electrical And Electronics Engineers, 2003).

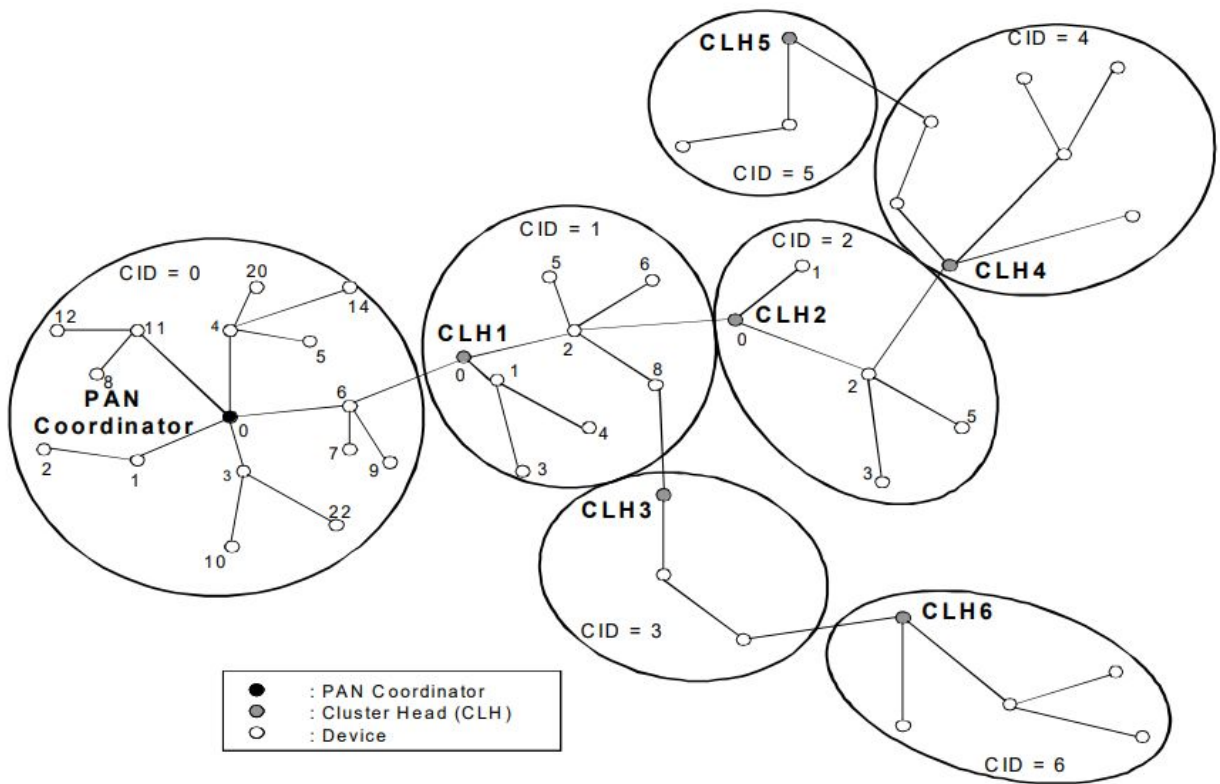


Fig I.C.1.e Cluster tree network (Institute Of Electrical And Electronics Engineers, 2003).

2. MQTT For Sensor Networks (MQTT-SN).

MQTT-SN is based on the MQTT network protocol but aimed at Wireless Sensor Networks. The protocol is meant to be compatible with any networking technology which allows a bi-directional transfer of data (upload/download) between nodes and a gateway. (Stanford-Clark and Truong, 2013)

The protocol uses a system of publish and subscribe messages to manage the transactions between publishers (end-device sending data), brokers (coordinators managing transactions) and subscribers (device sending a request for data to publish). (Hunkeler, Truong and Stanford-Clark, 2019)

The strong points of the MQTT-SN protocol are:

- Not standard specific which allows it to function easily with various communications technologies within the same network.
- It is a very lightweight and flexible protocol to implement and relatively power efficient.
- The protocol uses topics (synonymous to sub-applications within the network) rather than addresses to communicate with devices in the network thus creating a protective layer over the network architecture.

(Hunkeler, Truong and Stanford-Clark, 2019)

3. Microchip MiWi.

MiWi is a networking protocol stack developed by Microchip based on the IEEE 802.15.4 standard targeted at LR WPANs. The software stack provides applications to create, start, join and route data within networks.

MiWi offers network safety options in terms of transmitting messages and adding devices to the network. It also has the interesting features of “Over The Air Upgrade” to allow software updates for devices to be applied wirelessly within the network. (Microchip, 2019)

After contacting the customer support at Microchip, it is clear that although the software stack is open a lot of work is required to adapt it to third party platforms. The stack has been developed to be used with Microchip’s own SAMR21 and SAMR30 development kits.

4. ZigBee.

ZigBee is a networking protocol and software stack developed by the members of the ZigBee Alliance. It is aimed at radio frequency application necessitating low data rate, long battery life and secure networking. The software stack sits on top of the basic layers presented in the IEEE 802.15.4 (PHY and MAC). It provides the Network and the Application layers to offer a framework for more complex applications. (Wang, Jiang and Zhang, 2019)

ZigBee can offer management for self healing networks with up to 65,000 devices (ZigBee Alliance, 2019).

Another advantage of ZigBee is their customised software stacks which offer all round specific or complementary solutions such as:

- JupiterMesh - aimed at robust low data Industrial IoT applications (ZigBee Alliance, 2016a).
- Smart Energy - for monitoring, controlling, informing and automating the delivery and use of utilities (ZigBee Alliance, 2016b).
- Dotdot - a language to allow devices to communicate and interface more efficiently on a network (ZigBee Alliance, 2017a).
- Green Power - aimed at providing battery-less devices by harvesting energy in the device surroundings (ZigBee Alliance, 2017b)

III. Product Design Specifications.

Main functions: The platform should measure oxygen and carbon dioxide levels ($\pm 3\%$ accuracy), temperature and pressure within an environment fairly rapidly (below 30s). These sensor readings should be transferred wirelessly to a hub platform.

Networking aspects: The various sensor platforms should be able to receive and transmit data from another to relay the information to the hub platform from up to 100 meters. Ideally, the network should be organised as a mesh network allowing increased reliability. The hub platform will be connected to the internet/cloud via WiFi.

Environmental Considerations: The sensor platforms, being placed outdoors, will be required to offer protection against dust and water infiltration. An IP rating of 65 should offer adequate protection.

Energy Considerations: The system implemented off the grid will need to rely on batteries. The embedded platform and sensors will be required to operate on low voltage. Efforts will need to be made to offer an energy efficient system ensuring longevity and low maintenance.

IV. Concept Development and Prototyping.

A. Components Description.

This section will give an overview of the components used to create the wireless sensor network. The information presented is not meant as an exhaustive description but rather highlighting relevant elements justifying the choice of the component.

1. Arduino Microcontroller Platforms.

Arduino breakout boards are, usually AVR-based, rapid prototyping platforms. According to Hughes (2016) and Williamson (2014), AVR microcontrollers provide a multipurpose solution in one package including: a CPU, timers, counters, serial interface, analog to digital converters, analog comparators, interrupts and digital I/O ports. They are also relatively low power and offer various levels of voltage outputs. Arduinos are easily connected to a computer via USB. They are programmed using the Arduino language derived from C/C++ (both also usable to programme applications). They are supported by a large and active community with many example projects available. They are also relatively affordable. All of this makes them an ideal prototyping platform.

Two different types of Arduinos are used in this application, an Arduino Nano Every (Atmel megaAVR microcontroller) and an Arduino MKR WiFi 1010 (SAMD21 microcontroller).

Sensor Node - Arduino Nano Every:

The Arduino Nano Every has four USART serial ports and a Serial Peripheral Interface (SPI) which allow for the connection of multiple peripherals (Sensors and RF module in this application) to the board (Microchip, 2018). It can function with a power source as low as 5V with power outputs of 5 and 3.3V. It offers 50% more programme memory and twice as much RAM than the original Arduino Uno . The board has a very small footprint which makes it an ideal candidate for breadboard prototyping.(Arduino, 2019c)

Gateway - Arduino MKR WiFi 1010:

The Arduino MKR WiFi 1010 offers the possibility to connect easily to the internet through WiFi. It offers the possibility to connect the RF module thanks to the availability of an SPI port. Moreover, as the Nano Every, it has a low operating voltage at 3.3V but also provides output voltages of 3.3 and 5V. (Arduino, 2020)

2. RF Transceiver MRF24J40MA.

The MRF24J40MA is an affordable (generally below £9) radio frequency transceiver module manufactured by Microchip around its MRF24J40 chip. It complies with the IEEE 802.15.4 standard and supports ZigBee, MiWi, MiW P2P and other proprietary wireless networking protocols.

The module operates at low voltages between 2.4 and 3.6V with low current consumption from 19mA in reception mode and 23mA when transmitting with current as low as 2 μ A at sleep.

The transceiver communicates on an ISM Band 2.405-2.48 GHz offering 16 narrow-band radio frequency channels 5MHz apart using 2 MHz bandwidth. (Microchip, 2008a)

Below can be found a description and basic functional logic of the RF Transceiver functionalities. The module is programmed and operated through the module control registers using the SPI port of the Arduino. Software implementation is discussed in section “C.Application Software”.

Initialisation - This step is performed once when the device is powered. It configures the modules to allow it to perform its basic function. Control registers are set in sequence (Default and/or recommended values are given in the MRF24j40 datasheet):

1. Perform a reset - Can be achieved in four ways (Described in function below).
2. Enable FIFO method (FIFOEN bit) and the Transmitter Enable On Time Symbol bits (TXONTS) in the Power Amplifier Control 2 register
3. Initialise the VCO Stabilization Period bits (RFSTBL) in the Tx Stabilization register.
4. Initialise the RF optimize control bits (RFOPT) in the RF control 0 register.
5. Initialise the VCO Optimize Control bits (VCOOPT) in the RF control 1 register.
6. Enable Phase-Locked Loop (PLLEN) in the RF control 2 register to allow RF transmission or reception.
7. Set the TX Filter Control bit (TXFIL) and 20 MHz Clock Recovery Control bits (20MRECVR) in the RF control 6 register.
8. Configure the Sleep Clock Selection bits (SLPCLKSEL) in the RF control 7 register.
9. Set the VCO Control bit (RFVCO) in the RF control 8 register.
10. Disable the clock out (CLKOUTEN) and Sleep Clock Divisor bits (SLPCLKDIV) in the Sleep Clock Control 1 register.
11. Set Clear Channel Assessment (CCA) Mode bits to Energy Detection (ED) in the Baseband 2 register
12. Set CCA ED threshold (CCAEDTH) in the energy detection threshold for cca register.

13. Set RSSI calculation mode and append value to RXFIFO in the Baseband 6 register.
14. Enable interrupts - Described in function below.
15. Set channel - Described in function below.
16. Set transmitter power in RF control 3 register.
17. Reset RF state machine (RFCTL) in the RF mode control register.
18. Allow some delay for the main oscillator to stabilise before transmission or reception.

(Microchip, 2008b)

Reset (RST) - 4 Types of Resets are available on the MRF24J40 chip.

1. At power on, the chip automatically resets all the registers.
2. The module has a reset pin which allows for the microcontroller to reset the registers by setting the pin to low.
3. A software reset can be performed using the Software Reset register (SOFT_RST). It allows to reset the baseband, power management and/or MAC circuitry by setting the relevant bits.
4. An RF State Machine Reset can be performed by setting the RF Reset bit (RFRST) of the RF mode control register (RFCTL).

(Microchip, 2008b)

Interrupts (INT) - The purpose of the interrupt functionalities is for the transceiver to signal to the microcontroller an event has occurred. Those events can be configured using the Interrupt Control register (INTCON) and setting the relevant bits to low for instance:

RXIE: RX FIFO Reception Interrupt Enable bit.

TXNIE: TX Normal FIFO Transmission Interrupt Enable bit.

Interrupt Logic Sequence:

1. When an event occurs in the RF module, the related bit will be set in the interrupt status register (INTSTAT).
2. INTCON (as an inverted signal) and INTSTAT are then compared to confirm and trigger the event (high).
3. The interrupt logic circuit then compares the event to the Interrupt Edge Polarity bit (INTEDGE), set by default to a falling edge in the Sleep clock control 0 register (SLPCON0). The INTEDGE signal is inverted and compared to the event in a XOR gate. This ensures the transceiver interrupt output matches the INT pin logic of the microcontroller (triggered when low).
4. Once INTSTAT is read the bits in the register are reset (INT pin set back to high).

(Microchip, 2008b)

Channel Selection - This function configures the RF module to operate on one of the 16 frequencies available on the 2.4GHz band. The channel is selected by setting the channel bits to the proper value (See table V.A.2.a) in the RF control 0 register. Once the channel is set it is necessary to realise a Reset of the RF state machine to allow the module to calibrate properly. It is worth mentioning the channel needs to be the same for all nodes and coordinators on the same network.

Table V.A.2.a: *Channel Selection RFCON0 (0x200) register setting* (Microchip, 2008b).

Channel Number	Frequency	Set Value
11	2.405 GHz	0x03
12	2.410 GHz	0x13
13	2.415 GHz	0x23
14	2.420 GHz	0x33
15	2.425 GHz	0x43
16	2.430 GHz	0x53
17	2.435 GHz	0x63
18	2.440 GHz	0x73
19	2.445 GHz	0x83
20	2.450 GHz	0x93
21	2.455 GHz	0xA3
22	2.460 GHz	0xB3
23	2.465 GHz	0xC3
24	2.470 GHz	0xD3
25	2.475 GHz	0xE3
26	2.480 GHz	0xF3

Transmission - Can be realised through various methods using different types of frames and transmission (TX) FIFOs long address memory spaces as defined in the standard 802.15.14. For this application, a non beacon-enabled network, only the TX Normal FIFO is required (addresses 0x000 to 0x07F representing 128 bytes of memory space) to store the different frames and create a packet to transmit.

The TX FIFOs general structure is composed of four fields:

Header Length - The length in bytes of the MAC Header (MRH).

Frame Length - The MAC Header and payload length.

Header - Containing the MAC Header information.

Payload - Containing the data payload.

Sequence to send a packet using the TX Normal FIFO:

1. The TX Normal FIFO needs to be loaded with a packet structure as presented in figure V.A.2.b below.

Bytes	1	1	m	n
Packet Structure	Header Length (m)	Frame Length (m + n)	Header	Payload
TX Normal FIFO Memory Address	0x000	0x001	$0x002 - (0x002 + m - 1)$	$(0x002 + m) - (0x002 + m + n - 1)$

Fig. V.A.2.b: TX Normal FIFO Format.

2. From then, the Synchronization Header (SHR), PHY Header (PHR) and Frame Check Sequence (FCS), are appended to the previous packet structure as required to insure the packet follows the recommendations made in the IEEE 802.15.4 standard for packet transmission.
3. An automatic acknowledgement of reception can be set for each packet sent. This is achieved by setting the acknowledgement request bit in the frame

control section of the MRH when loading to TX Normal FIFO. Additionally, it is necessary to set high the TX Normal FIFO Acknowledgement Request bit (TXNACKREQ) in the Transmit Normal FIFO Control register (TXNCON).

4. The Transmit Frame in TX Normal FIFO (TXNTRIG) bit from TXNCON register is then set to high to transmit. The CSMA-CA algorithm will then take care to retransmit until the acknowledgement of reception is received.
5. The TX Normal FIFO Release Interrupt bit (TXNIF) of the INTSTAT Control register (i.e interrupts management) will set to high to indicate a acknowledgement of reception is received.

(Microchip, 2008b)

Reception - In reception mode a device monitors signals in the air and looks for any packets with an IEEE 802.15.4 valid packet preamble. This means a packet starting with a Synchronization Header (SHR) containing the preamble sequence and Start-of-Frame Delimiter (SFD) fields. The purpose is for the RF module to recognise if the packet can be read.

The packet is then unpacked to perform a Cyclic Redundancy Check (CRC). This verifies if the packet information matches the data extracted.

The packet is then accepted or rejected according to the reception mode and frame filter.

There are three reception modes which can be configured using the Receive Mac Control register (RXMCR):

Normal Mode - Only accepts packets with a good CRC and compliant with the IEEE 802.15.4 specifications.

Error Mode - Accepts packets with good or bad CRC (Relevant mode for debugging).

Promiscuous Mode - Accepts all packets with a good CRC.

There are four frame filters which can be configured in the Receive Fifo Flush register (RXFLUSH): All Frames, Command Only, Data Only and Beacon Only. This will define which frames are passed on to the RX FIFO buffer.

At this stage, if an acknowledgement request was configured into the packet, TXMAC circuitry of the RF module will execute the request without the microcontroller having to process the request.

As the frames are passed onto the RXFIFO a receive INT signal will be sent to the microcontroller (as described in the Interrupt function) to read the content of the buffer through its SPI port (described in “B.Components Interfacing Methods”).

Reading RX FIFO buffer sequence:

1. Receive INT.
2. Disable the microcontroller interrupts.
3. Set the RX Decode Inversion bit (RXDECINV) to high in the BBREG1 Control register. This will avoid reading any other packet in the air.
4. Obtain the RX FIFO length by reading the first address memory space (0x300).
5. Read RXFIFO addresses, 0x301 through (0x300 + Frame Length + 2) to extract the data, RSSI and LQI values.
6. Reactivate the packet reception by setting RXDECINV back to low.
7. Re-enable the microcontroller INT.

(Microchip, 2008b)

Received Signal Strength Indicator (RSSI)/Energy Detection (ED) and Link Quality Indication (LQI).

RSSI/ED is an important indicator because it can be used to detect if a device is idle or busy according to pre-configured energy levels (see Initialisation function).

It can be obtained in two different ways:

RSSI Mode 1 - By setting the RSSIMODE1 bit in the BBREG6 register will initiate the calculation. Once the RSSI Ready Signal bit (RSSIRDY) is set to high the average RSSI power value is ready to be read in the RSSI register.

RSSI Mode 2 - Will append the RSSI value to the RX FIFO by setting high the RSSIMODE2 bit in the BBREG6 register.

In both modes, the RSSI returned is an 8-bit value between 0 and 255. Figure V.A.2.c shows the relation between the received power in dBm and the RSSI value.

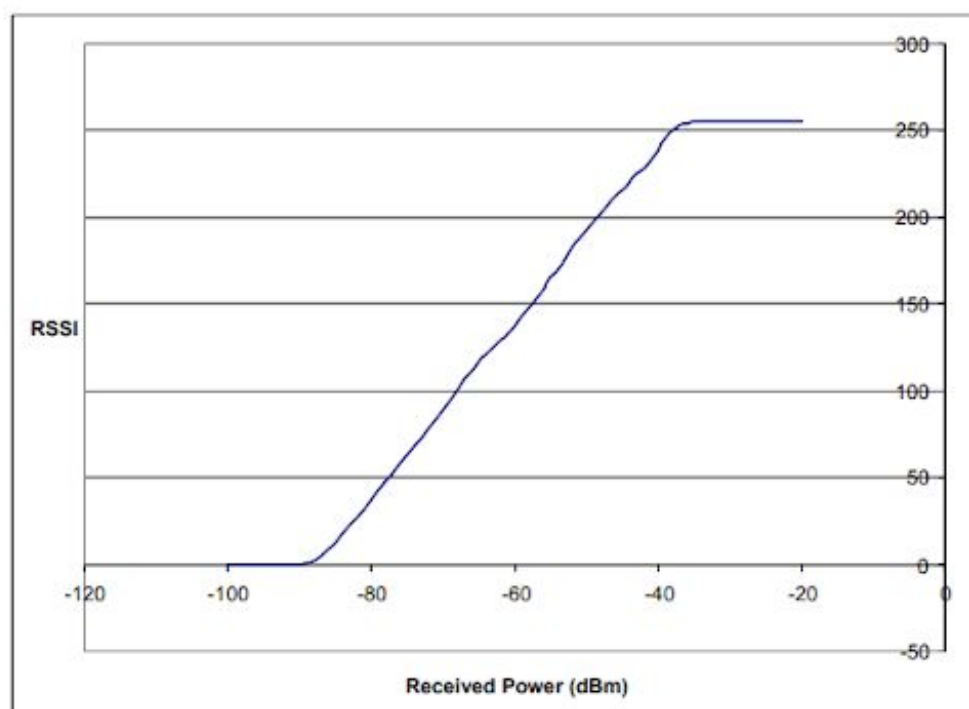


Fig V.A.2.c RSSI vs. RECEIVED POWER (dBm) (Microchip, 2008b).

The LQI demonstrates the strength or quality of a received packet. A packet received with a high RSSI and LQI will be of better value. As for the RSSI, the LQI is a value between 0 (very low link quality) and 255 (high quality of link). The returned LQI value is automatically appended to the RX FIFO for each received packet.

(Microchip, 2008b)

3. Carbon Dioxide Sensor: CoZIR-AH-5000.

The CozIR is a low power Solid-state Non-Dispersive InfraRed (NDIR) CO₂ sensor. It requires about 50 times less power than an ordinary NDIR sensor which makes it an ideal candidate for WSN and battery powered applications with an ideal operating voltage of 3.3V . It can return CO₂ concentration readings (ranges from 0-2000ppm, 0-5000ppm and 0-1%) as well as temperature and relative humidity (RH%). The measurements are obtained digitally through a UART port on the microcontroller (as described in “B.Components Interfacing Methods”). This allows to set the sensor to one of the three operating modes and perform measurements using a set of commands.

CoZIR sensor - Operating Modes:

The mode can be set by using the command K as described in the following section. Only one mode can be selected and operate at once.

Mode 0 Command - This mode can not obtain any measurements. Its purpose is to obtain information relative to the sensor such as: configuration information, firmware version and serial number.

Mode 1 Streaming - In this operating configuration, the sensor will return a measurement every 0.5 seconds continuously according to the entered request command.

Mode 2 Polling - In this mode, the sensor will return a value only when a request command is entered. The sensor continues operating in the background but only transmits when requested by the microcontroller.

(GSS, 2018)

Relevant Commands:

The commands below are sent to the sensor by the microcontroller through the UART (more information on the software implementation later in the report). The sensor will send a response to the request. Only the basic commands relevant to the application are presented here. More advanced and complementary commands can be found in the “GSS Sensor User Guide”.

Mode Selection - Command: “K #\r\n” where “#” is a number and “\r\n” indicates the end of the command. This allows the user to set the selected operating mode. For example “K 2\r\n” to set the sensor to Polling Mode. The sensor then returns “K 00002\r\n” to confirm the operating mode is set to Mode 2.

Calibration using fresh air - Command: “G\r\n”. Placing the sensor in a fresh air environment and sending the command allows it to automatically calibrate the zero point assuming the sensor is in a 450ppm CO₂ environment (which is an accepted average value for fresh air). Other more accurate calibration techniques, requiring defined levels of gases in a closed environment, are available in the Sensor User Guide. The sensor will return “G 33000\r\n” (the number is variable) to confirm.

Humidity Measurement - Command: “H\r\n”. The command will return the humidity measurement from the sensor. The returned value will have to be divided by 10 to obtain the relative humidity (%RH). Returned value example: “H 00551\r\n”. (i.e: 55.1%)

Temperature Measurement - Command: “T\r\n”. The command will return the temperature measurement from the sensor. To obtain the temperature value, a 1000 will have to be subtracted from the reading. This value will then be divided by 10 to obtain a value in degree celsius . Returned value example: “T 01224\r\n.” (i.e: 22.4°C)

CO2 Measurement - Command: "Z\r\n". The command will return the carbon dioxide measurement from the sensor in parts per million (ppm). The returned value will have to be divided by 10 to obtain the relative humidity (%RH). Returned value example: "Z 00512\r\n" (i.e: 512 ppm) if digitally filtered or "z 00512\r\n" unfiltered. (Raw data reading returns filtered and unfiltered in the same string)

(GSS, 2018)

4. Oxygen Sensor: LuminOx Optical Oxygen.

The LuminOx is a low-powered factory calibrated oxygen sensor which can produce measurements for: ppO2 (ppO2 in mbar), Temperature (°C), Pressure (mbar) , O2 (in %). It operates between 4.5 to 5.5V which makes it appropriate for battery powered applications. All the signal conditioning is realised within the sensor, therefore it only requires a 3.3V UART interface to communicate with a microcontroller. As for the CoZIR, the sensor can be operated in 3 different modes and functions using a set of commands. (SST, 2016)

LuminOx sensor - Operating Modes:

Mode 0 Stream Mode - This mode, similarly to Streaming in the CoZIR, will return every second measurements for ppO2 (ppO2 in mbar), Temperature (°C), Pressure (mbar) , O2 (in %) and sensor status.

Mode 1 Poll Mode - Similarly to Polling in the CoZIR, the sensor will return values and information according to the commands entered.

Mode 2 Off Mode - Here, the sensor stops making measurements to save power (less than 6mA consumed).

(SST, 2016)

Relevant Commands:

The process for sending commands is the same as for the CoZIR sensor previously presented.

Mode Selection - Command: `"M #\r\n"`. Where `"#"` is the output mode to select: 0 = Stream, 1 = Poll or 2 = Off. This will set the sensor to one of the operating modes. The sensor will confirm by returning `"M xx\r\n"` (`"xx"` being 0 followed by the mode number).

O2 Measurement - Command: `"O\r\n"`. The sensor will return the ppO2 reading in ppO2 in mbar in the format `"O xxxx.x\r\n"` (`"xxxx.x"` being the numerical value).

O2 Measurement - Command: `"%\r\n"`. This command will return an O2 reading in percentage with a returned message as: `"% xxx.xx\r\n"` (`"xxx.xx"` being a percentage).

Temperature Measurement - Command: `"T\r\n"`. The command will return a sensor internal temperature in degree celsius. Message returned: `"T yxx.x\r\n"` (`"y"` being a +/- and `"xx"` a numerical value in °C).

Barometric Pressure Measurement - Command: `"P\r\n"`. The request returns the current barometric pressure value in millibar with a returned message as: `"P xxxx\r\n"` (`"xxxx"` is a numerical value in mbar)

Sensor Status - Command: `"e\r\n"`. This command will return two types of messages:

First message: `"e 0000\r\n"` meaning the sensor is functional and operating correctly.

Second message: `"e xxxx\r\n"` indicating an issue or error. Common error messages:

`"E 00\r\n"` - RS232 Receiver Overflow.

`"E 01\r\n"` - Invalid Command.

`"E 02\r\n"` - Invalid Frame.

`"E 03\r\n"` - Invalid Argument.

All Measurements and Sensor Status - Command: "A\r\n". This command will return all the results of the previous request commands as one string. It is especially useful for the Stream Mode as it offers a more compact command.

(SST, 2016)

5. Bi-Directional Logic Level Converter.

The components (RF Transceiver and sensors) interfacing with the Arduino Nano operate using 3.3V signals when the digital output of the Arduino operates at 5V. Margolis (2018) recommends using a logic-level converter to protect the devices. The bi-directional logic-level converter allows for the 5V logic signal coming from the Arduino to be converted to a 3.3V signal allowable by the device. This will also ensure the signal sent from the devices will match the 5V digital input level of the Arduino.

6. Power Supply and Sleep Mode Considerations.

Arduino Boards Power Supply:

Although most of the operation of the Arduinos, during prototyping, was done whilst being powered through the USB port of the PC, it is possible to operate them using battery packs. The Arduino Nano Every offers more flexibility for batteries with an operating voltage of 5V but an input voltage limit of 21V (Arduino, 2019c). Essentially, the Nano Every can be powered from common batteries providing at least 5V. For this project, 9V 200mAh NiMH rechargeable batteries were selected because of their wide availability and ease of use. They will provide enough power to match the consumption of the Arduino Nano Every and Peripherals.

The Arduino MKR WiFi 1010 can only receive a voltage of 5V and operates at 3.3V. The Arduino MKR has the advantage of integrating a Li-Po charging circuit which allows it to run from and charge such batteries when USB powered. It is recommended to use a "Li-Po Single Cell, 3.7V, 1024mAh Minimum" (Arduino, 2019c).

Power Consumption and Sleep Mode:

A WSN being possibly implemented off the grid and/or remote locations, batteries are the only option available. An Arduino running from a 9V battery is likely to run only for a couple of weeks if constantly powered (Margolis, 2018). Therefore, it is primordial to pay attention to power consumption and put in place low power usage strategies.

The first option is to disabled or physically remove unused peripherals of the Arduino not used by the application. The second option is to only enable those peripherals when they are required by the application using software libraries. A third option is to reduce the clock speed provided it does not impact the running of the application. Ideally all of those options are explored to put in place a global power management solution. (SparkFun Electronics, 2020)

Williamson (2014) explains AVR microcontrollers offer various sleep modes which can be indefinite, timed or terminated by interrupts. Each microcontroller will have its specific sleep modes detailed in its datasheet (Usually under “Power Management and Sleep Modes”).

Essentially there are three different main sleep modes (some libraries and/or MCUs offer other options):

Idle mode - This is the simplest sleep mode. Only the CPU clock is turned off essentially disabling the chip whilst waiting for an interrupt (hardware or timer) to wake it. All other functionalities (timer, ADC and I/O) are running in the background.

ADC noise reduction mode - This mode shuts the CPU and I/O clocks whilst keeping the ADC on waiting for a flag or interrupt to trigger the wake up sequence.

Power down mode - This is the deepest sleep mode available. All the clocks and peripherals are turned off waiting for an interrupt to wake the system up.

(Williamson, 2014)

It is worth noting the deeper the sleep mode the longer the wake up process will be hence the more power will be consumed over an application cycle.

Power Consumption Estimation:

Calculating accurate power consumption the wake and sleep phases of the applications is fairly complex because of the various elements influencing how much current is being used. Choudhuri (2017) offers a simple method to determine an estimation of the power requirements of a standalone system from information available from datasheets. Margolis (2018) indicates that a board normally consuming 15 mA could in theory be brought down to 0.001 mA using various power reductions and sleep methods. To simplify the calculations, it will be assumed the sleep mode for the microcontroller will bring the current consumption down by 80% from normal operation.

Sensor Node - Calculations:

Table V.A.6.a: *Sensor Node Main Electronic Components Power Consumption.*

Equipment	Normal	Sleep
Arduino Nano Every	19 mA	3.8 mA
LuminOx Sensor	7.5 mA	6 mA
CoZIR Sensor	1.5 mA	0.5 mA
MRF24J40MA (Transmission)	23 mA	0.002 mA

Measurement and Transmission Phase - The calculations are realised on the basis of a 10 seconds measurement cycle to allow for the wake up phase of the microcontroller and sensors to settle with four measurements realised per day. The transmission calculation is based on a 2 seconds cycle.

Daily Measurements Calculation:

$$(Arduino\ Nano\ Every + LuminOx + CoZIR) \times 10s\ in\ hour \times 4\ measurements$$

$$(19 + 7.5 + 1.5) \times 0.003 \times 4 = 0.336\ mAh\ a\ day$$

Daily Transmissions Calculations:

$$MRF24j40MA\ in\ Tx\ Mode \times 2s\ in\ hour \times 4$$

$$23 \times 0.0005 \times 4 = 0.0115\ mAh\ a\ day$$

Daily Operations Calculations:

$$Daily\ Measurements + Daily\ Transmissions$$

$$0.336\ mAh + 0.0115\ mAh = 0.3475\ mAh\ a\ day$$

Sleep Phase - Calculations are realised assuming the system will be asleep for 23.96 hours per day and that a power consumption drop of 80% is achieved for the microcontroller.

Daily Sleep Mode Calculations (Components at Sleep) :

$$(Arduino\ Nano\ Every + LuminOx + CoZIR + MRF24J40MA) \times 23.96\ hours$$

$$(3.8 + 6 + 0.5 + 0.002) \times 23.96 = 246.8\ mAh\ per\ day$$

Therefore, the estimated power consumption for one of the sensor nodes is too high for the proposed battery at : **247.15 mAh per day**. (Results discussed in “VI.Results and Prototype Evaluation”)

Gateway Coordinator - Calculations:

Table V.A.6.b: *Gateway Coordinator Main Electronic Components Power Consumption.*

Equipment	Normal	Sleep
Arduino MKR WiFi 1010 (CPU and Peripherals)	5.32 mA	1.06 mA
u-blox NINA-W102 WiFi Module (Transmission)	190 mA	0.008 mA
MRF24J40MA (reception)	19 mA	0.002 mA

Receiving and Transfer through WiFi - The calculations are based on receiving and retransmitting four measurements a day from one sensor node platform.

Daily *Measurements* Reception Calculations:

$(\text{Arduino MKR} + \text{uBlox Module} + \text{MRF24J40MA}) \times 10s \text{ in hour} \times 4 \text{ measurements}$
 $(5.32 + 190 + 19) \times 0.003 \times 4 = 2.572 \text{ mAh a day.}$

Sleep Phase - Calculations use the 80% power reduction estimation for the microcontroller with a sleep period of 23.96 hours.

Daily Sleep Mode Calculations (Components at Sleep) :

$(\text{Arduino MKR} + \text{uBlox Module} + \text{MRF24J40MA}) \times 23.96 \text{ hours}$
 $(1.06 + 0.008 + 0.002) \times 23.96 = 25.64 \text{ mAh a day}$

Therefore, the estimated power consumption for the Gateway platform (with the proposed battery) would last an average of 35 days at: **29.112 mAh per day.**
(Results discussed in “VI.Results and Prototype Evaluation”)

B. Components Interfacing Methods.

This section will give a short explanation about the two methods used by the microcontroller to communicate with the sensors and RF transceiver to obtain and transmit the data as well as the system configuration. More information will be available in the following section about the software implementation.

Both methods fall within the serial communication. It has been used since the inception of microcontroller based computers under the RS-232 standard and supplanted by the Universal Serial Bus for PC peripherals. Although, basic serial communication is still in use in microcontroller applications because of their simplicity to implement and reliability. They required at least two cables Rx to receive and Tx to transmit information and connected as described in the figure V.B.a below. (Langbridge, 2015)

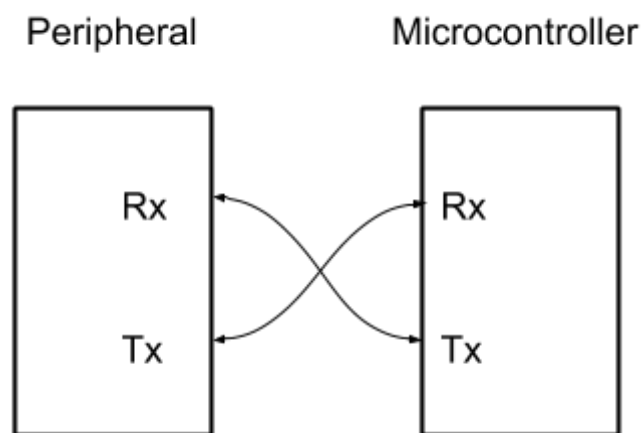


Fig V.B.a Serial Communication Port.

1. Universal Asynchronous Receiver/Transmitter (UART).

The UART is a hardware interface allowing the microprocessor to easily communicate with peripherals using the serial protocol. On AVR microcontrollers the port is actually USART. This means the interface can be used asynchronously and synchronously (with a clock signal) (Hughes, 2016).

As described in the components description, the sensors function using the UART port. According to the mode selected and sent requests, the sensors will return

measured values. Hence the importance of understanding the Rx/Tx transmission to send and receive data.

To communicate, the microcontroller and peripheral interfaces need to share a common configuration including: baud rate and serial packet structure.

The baud rate is a speed rather than the amount of bits sent per second. It can vary from 300 to 115,200. The usual baud rate for an arduino is 9,600. It is important both parties are configured to transmit and receive at the same baud rate or else the printed data will be unreadable. Since the communication does not use a clock and runs continuously, the serial packet structure gives an indication as to when a character starts and ends. The RS-232 indicates the packet frame should be at least 10 bits and set as in Fig V.B.1.a below. (Langbridge, 2015)

Start	Data	Stop
1 bit	5-9 bits (8 bits for ASCII)	1-2 bits

Fig V.B.1.a RS-232 Serial Packet Basic Structure.

Essentially, the bits are set as low (0V) or high (3.3 or 5V according to peripheral specification) to encode characters using binary. Williamson (2014), demonstrates how to encode the integers 9 and 10 as in fig. V.B.1.b below.

Start	Data: "9"	Stop	Start	Data: "10"	Stop
0	"10010000"	1	0	"01010000"	1

Fig V.B.1.b Packet Structure for "9" and "10".

It is important to notice the binary sequence is inverted, the most significant bit (MSB) is transmitted first and finally the least significant bit (LSB) closes the data sequence. Information about how to cope with this situation will be available in "C. Application Software".

2. Serial Peripheral Interface (SPI).

The SPI bus is similar to UART in the sense that it uses two data lines MISO (Master In - Slave Out) and MOSI (Master Out - Slave In). The first difference is that SPI uses two more wires than UART, Slave Select (also referred to as Chip Select) and SCLK (or SCL). Thanks to the SS/SCLK combination and the 2 way communication happening at the same time, transfers between the “slave device” (in this application the RF Transceiver) and “master” (the Arduino) are much faster than with UART. Another advantage of the SPI is the possibility of connecting several slaves on the same SPI port as demonstrated in Figure V.B.2.a. The slave selection is done by pulling the SS pin to low for the desired peripheral hence high when not selected (Williamson, 2014)

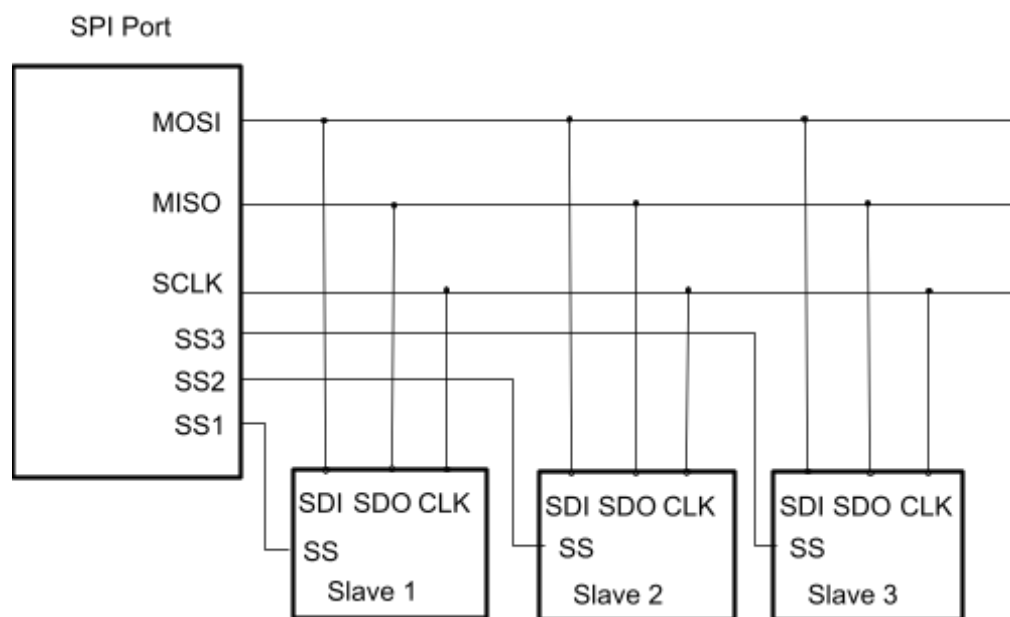


Figure V.B.2.a Serial Peripheral Interface with 3 Slave Devices.

To function the SPI requires three parameters: the Bit Order, the Clock Mode and the Clock Speed. The bit order defines which bit is sent first MSB or LSB. The clock mode defines the clock polarity and phase (as described in table V.B.2.b). Finally the clock speed which is usually in the range 1-10 MHz (16MHz by default on Arduino) but which allows even faster speed. (Langbridge, 2015)

Table V.B.2.b. The Different SPI Clock Modes (Langbridge, 2015)

Mode	CPOL	CPHA	Effect
SPI_MODE0	0	0	Clock base 0, capture on rising, propagation on falling.
SPI_MODE1	0	1	Clock base 0, capture on falling, propagation on rising.
SPI_MODE2	1	0	Clock base 1, capture on falling, propagation on rising.
SPI_MODE3	1	1	Clock base 1, capture on rising, propagation on falling.

When transferring information from or to the peripheral the SPI uses 8-bit shift registers (1 byte of data).

Communication sequence between a Master and Slave in a parallel to serial conversion:

1. Both Master and Slave set their shift register to the required bit sequence. The Master will send a command and the Slave will answer a previous request.
2. At each turn of the clock the registers will shift one bit. The Slave receives a bit from the Master and reciprocally the Master receives one bit from the Slave.
3. At 8 turns of the clock, messages will have been exchanged and the respective registers will be reset accordingly.

It is worth noting a request command from the Master could be 8-bit but a message from the Slave could be a 16-bit answer. In this case, the Master will keep passing 0 at each turn of the clock until all bits from the Slave have been received. The transaction will then be finished. (Langbridge, 2014)

C. Application Software.

This section gives an overview of the implementation of the application functionalities onto the Arduino microcontroller. A succinct description of libraries and low level functions are given followed by run through the main application: obtaining the data from the CoZIR sensor, packaging and transmitting the measurement to another platform. The application codes for the Sensor Node and Gateway Coordinator are available in appendix A (Sensor Node Application Code) and B (Sensor Node Application Code). Due to components availability and technical issues with the Arduino MKR WiFi1010, the software presented in this section only uses the CoZIR sensor and two Arduino Nano Every (one sensor node and one gateway coordinator) to demonstrate the principles of the application.

1. Libraries and Read/Write from/to Control Registers.

This section introduces the principal libraries used to allow the different components to interact with one another. It also presents the basic functions required for the microcontroller to interact with the RF Transceiver control registers.

Libraries Used in the Application.

Software Serial Library - This is an official Arduino library which allows the microcontroller to utilise any digital pins, to use as Rx and Tx, with its UART interface. Therefore it can provide multiple serial software ports on a single microcontroller. The library provides functions to simply configure, operate and read/write from serial ports. (Arduino, 2019a)

SPI Library - This is an official Arduino library which allows the microcontroller (master) to use SPI ports to communicate with SPI devices (slaves). The library provides functions to simply configure, operate and read/write data from and to the peripherals. (Arduino, 2019b)

CozIR Library (Appendix C) - This library was written by Rob Tillaart and Michael Hawthorne for the Arduino Community to easily integrate the CoZIR sensor into projects. The functions available are based on the functionalities described in the “GSS Sensor User Guide”: Configuring the serial port and sensor, setting the operating mode, sending a request and reading a measurement. (Tillaart and Hawthorne, 2018)

Mrf24j Library (Appendix D) - This library was written by Karl Palsson for the Arduino Community to use the MRF24j40MA Transceiver with Arduino boards. The library offers high level functions to configure, initiate and operate the transceiver as described in the MRF24J40 datasheet (and presented in V.A.2). Those functions are enabled by using the “read and write to a register” functions presented in the following section. The library uses the previously presented SPI library. (Palsson, 2019)

Interfacing with Control Registers.

Communicating with the RF module requires the microcontroller to be able to read and write data to the control registers. Moreover, the functions call for the possibility to read or write 1 byte (8 bits) or 2 bytes (16 bits) hence the requirement for “short” and “long” functions.

Reading data from a Control Register - The two following functions allow to read information stored in the RF Transceiver registers. As previously mentioned, the data transfer can be for a “short” or “long” address.

Read Short Function Explanation:

```
byte Mrf24j::read_short(byte address) { // address = register to be read.
    SPI.beginTransaction(spiSettings); // Start SPI with the defined SPI settings.
    digitalWrite(_pin_cs, LOW);        // Chip Select to low = Enable the Slave.
    SPI.transfer(address<<1 & 0b01111110); // 0 MSB for short, 0 LSB for read
    byte ret = SPI.transfer(0x00);      // ret is a variable holding the data read.
    digitalWrite(_pin_cs, HIGH);       // Chip Select to high = Disable the Slave.
    SPI.endTransaction();              // Shut the SPI connection.
    return ret;                        // return the data read.
}
```

Read Long Function Explanation:

```
byte Mrf24j::read_long(word address) { // word = 2 bytes (because long addressing)
    SPI.beginTransaction(spiSettings);
    digitalWrite(_pin_cs, LOW);
    byte ahigh = address >> 3; // 16 bit Address conversion high part.
    byte alow = address << 5;  // 16 bit Address conversion low part.
    SPI.transfer(0x80 | ahigh); // 0x80 - 1 for MSB = long. Call address high.
    SPI.transfer(alow);        // Call address low.
    byte ret = SPI.transfer(0);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction();
    return ret;
}
```

Writing data to a Control Register - The two following functions allow the microcontroller to write information to the RF Transceiver registers. As previously mentioned, the data transfer can be for a “short” or “long” address.

Write Short Function Explanation:

```
void Mrf24j::write_short(byte address, byte data) {    // Data to write in address.
    SPI.beginTransaction(spiSettings);
    digitalWrite(_pin_cs, LOW);
    SPI.transfer((address<<1 & 0b01111110) | 0x01); //MSB/LSB at 0 = short/write.
    SPI.transfer(data);                               // Write data in register.
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction();
}
```

Write Long Function Explanation:

```
void Mrf24j::write_long(word address, byte data) {
    SPI.beginTransaction(spiSettings);
    digitalWrite(_pin_cs, LOW);
    byte ahigh = address >> 3;
    byte alow = address << 5;
    SPI.transfer(0x80 | ahigh);           // 0x80 - 1 for MSB = long.
    SPI.transfer(alow | 0x10);           // 0x10 - 1 for LSB = write.
    SPI.transfer(data);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction();
}
```

2. Setup and Initialisation phase.

This phase of the application is similar for Sensor node and Gateway Coordinator. The only difference is the sensor aspect, the transceivers on each platform are both capable of sending and receiving packets hence the setup and initialisation being the same.

Sensor Setup and Initialisation:

1. Initialisation of the two related libraries: SoftwareSerial and coziR.
2. The Rx and Tx pins for the communication with the sensors are defined and passed on to the coziR library.
3. Relevant variables are created to store the readings at various stages of the reading: buffer and index, string holding the value post buffer extraction, the multiplier constant to convert to ppm, the variable to hold the final CO2 usable reading.
4. In the setup function, the serial connection is initiated (serial1 for this application). The operating mode is also set at this point (Polling Mode - most energy efficient mode).

MRF24J40MA Transceiver Setup and Initialisation:

1. Initialisation of the two related libraries: SPI and mrf24j (also initialises variables related to the transceiver operation: buffer, packet frames, flags and assigns the register addresses to a variable).
2. Pin assignment for: Reset, Chip Select, Interrupt and Clock. Those are then passed to the mrf24j library for initialisation of the SPI port (RST: output, CS: output, INT: input and SCK: output).
3. Time variables to control the transmission intervals are created: last_time and tx_interval.
4. The setup phase will perform a reset and initialisation (as described in section V.A.2 Reset and Initialisation functions). The initialisation is performed by writing the desired configuration to the control registers using the write functions previously described.

5. The sensor node and gateway are then set to the “promiscuous” reception mode (description available in V.A.2 Reception). Explanation for the mode selection available in “D. Activities undertaken during prototyping.”
6. The panID (Network ID) and address are set for the node. The network ID must be the same as the gateway coordinator to be recognised.

Interrupts Setup and Initialisation:

The interrupt is set up and initialised by calling the “attachInterrupt” function. It requires three arguments:

- digitalPinToInterrupt: setting the pin to an interrupt.
- Interrupt Service Routine (ISR): Tasks performed when the interrupt is triggered. Here the “interrupt_routine()”.
- Mode: defining when the interrupt is triggered (Here, the pin will go to low when triggered).

(Arduino, 2019c)

3. Obtaining Sensor Data.

Obtaining a sensor reading is done in two phases:

First, the sensor receives a command request (as described in V.A.3) sent through the Serial1. The sensor returns a value stored in the buffer until the ASCII value "0x0A" (new line) is returned.

Secondly, the data stored in the buffer is unpacked and converted using "unpack()". To do so, the function loops over the buffer until it hits the lower case "z" character (corresponding to the unfiltered value as previously described in V.A.3 CO2 Measurement). The numerical characters are then saved to an array (i.e: excluding "Z" and spaces). The characters saved in the array are ASCII characters. In ASCII, numerical values (0 to 9) are numbered 48 to 57 (<http://www.injsoft.se>, 2005), therefore 48 has to be subtracted from each value to get a numerical value. This array could directly be used as a message to send the string to the gateway since the conversion multiplier to apply for this sensor is 1 and no other processing is required. To verify the value and display it on the serial monitor, the value has to be converted to an integer using the "toInt()" string function to obtain the measurement as a number.

4. Packaging and Sending.

In the case the sensor used a different multiplier, the CO2 value would have to be converted back to an array using "itoa()". This function takes three arguments: integer value (co2 for this application), a string array to store the characters (here msg_send) and an integer base (decimal for this example). (cplusplus.com, 2000)

This value can then be passed to the send16() function of the mrf24j library. It takes two arguments: The target address and character string to be sent. The function is constrained to be executed under one second. "send16()" uses the "write long function" to fill in the different register addresses of TX FIFO (as described in V.A.2 Transmission) to build the packet frame with the required information. The auto acknowledgement is set before the packet transmission is triggered.

5. Receiving and Reading Data.

In order to receive an incoming message two functions are running at the same time first the interrupt routine, which will be triggered on the INT pin from the RF transceiver goes to low, and the “check_flags()” function which determines which routine to trigger according to the packet received normal message or packet reception confirmation.

First, once the INT is triggered the interrupt handler will determine whether to create a message reception or acknowledgment reception confirmation flag. The function then disables packet reception, and interrupts and starts getting the packet information from the relevant registers in the RF transceiver: frame length, data, LQI and RSSI stored in the relevant buffer using the “read long function” previously described.

In the meantime the check_flags() routine triggers the appropriate handle routine according to the flag and starts unpacking the various information from each buffer and printing them into the serial monitor.

Finally, the packet reception and interrupt are re-enabled for the next packet reception.

The process described for these two functions follows the logic described in “V.A.2 Reception”.

D. Activities undertaken during prototyping.

This section describes the activities undertaken and interesting points observed during the development of the sensor node and gateway. A large part of the activities are focused on fault finding on the RF Transceiver. Due to time constraints and technical difficulties with the Arduino MKR WiFi 1010, activities for the WiFi gateway are limited.

1. CoZIR Sensor Basic Measurement.

The CoZIR obtains the carbon dioxide measurements as previously described in section V.A.3. It requires four cables: two for the serial communication, to send request and received data, and two for power: 3.3V and ground. The sensor has total of ten pins with only four connected to the Arduino as follows:

Table V.D.1.b: *Arduino and CoZIR-AH-5000 Wiring.*

Arduino	CoZIR-A
GND	GND
3.3V	3V3
Tx (pin 1)	Rx
Rx (pin 0)	Tx

Table V.D.1.b: *CoZIR-AH-5000 Pin Layout (based on Fig V.D.1.b below) (GSS, 2018).*

Pin Name	Pin Number	Pin Number	Pin Name
Fresh Air Zero	10	9	Analogue Output
Nitrogen Zero	8	7	Sensor Tx (out)
GND	6	5	Sensor Rx (in)
GND	4	3	3V3
Not Connected	2	1	GND



Fig V.D.1.b CoZIR-AH-5000 connector pins.

The first test was realised on an Arduino Uno as demonstrated in Fig V.D.1.c. An example of the readings obtained can be found in Fig V.D.1.d. Measurements are returned at around one per second.



Fig V.D.1.c CozIR-AH-5000 Connected to Arduino Uno.

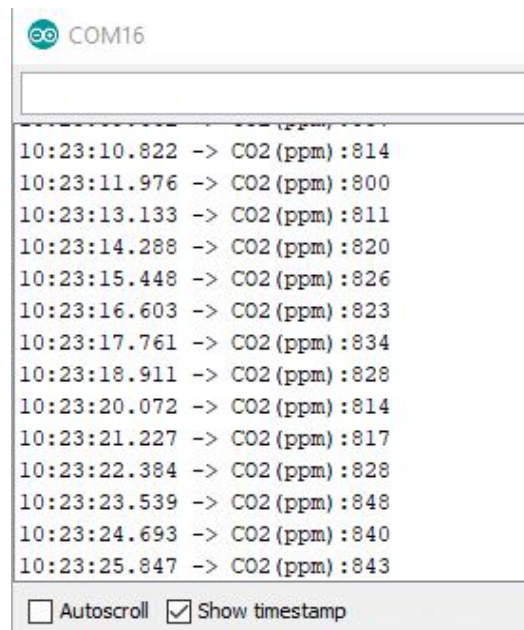


Fig V.D.1.d Serial Monitor CO2 Reading from CozIR sensor Output.

2. Integrating the MRF24J40MA Transceiver.

Most of the prototyping activities were spent on integrating the transceiver to the project, debugging and finding solutions to the telecommunication issues between the sensor node and gateway.

Assembly of the sensor node and gateways prototypes - The prototypes were assembled using breadboards and jumper wires. This is an easy and rapid method to test circuits. The Arduino Nano Every is delivered with its pin headers unsoldered to allow an easier integration into other printed circuit boards. The MRF24J40MA module is sold as a surface mount component. Both components had to be fitted with pin headers to allow them to be used on breadboards as shown in Fig V.D.2.a below.

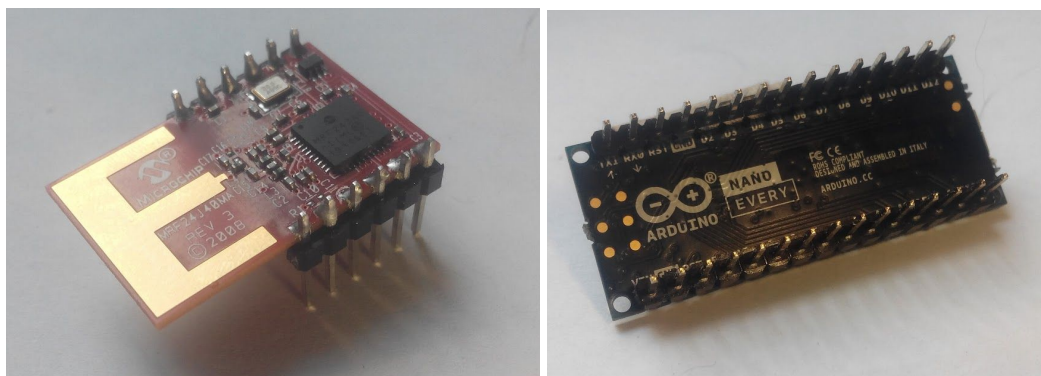


Fig V.D.2.a MRF24J40MA and Arduino Nano Every Pin Header Assembly.

Both components are mounted onto the breadboard and connected as described in Table V.D.2.a. Additional components were added: an LED to indicate the RF modules is being supplied with power, a bi-directional logic level converter and a decoupling capacitor (justification available in “Debugging Transceiver Communication Issues”). Sensor node and Gateway prototypes examples are available below: Fig V.D.2.b, Fig V.D.2.c and Fig V.D.2.d (Note: Additional pin headers were installed on the breadboard to allow easy access with measuring equipment).

Table V.D.2.a: *Arduino-MRF24J40MA Pin Connections* (Microchip, 2008a).

Arduino Pin Name	Pin Number	Pin Number	MRF24J40MA Pin Name
Ground	GND	1	GND
Reset (Digital pin Output)	6	2	Reset
Not Connected	x	3	Wake
INT 0 (Digital pin Input)	2	4	INT
MOSI	11	5	SDI
SCK	13	6	SCK
MISO	12	7	SDO
SS (Digital pin Output)	8	8	CS
Not Connected	x	9	NC
3V3 Supply	3.3V	10	Vin
Not Connected	x	11	GND
Not Connected	x	12	GND

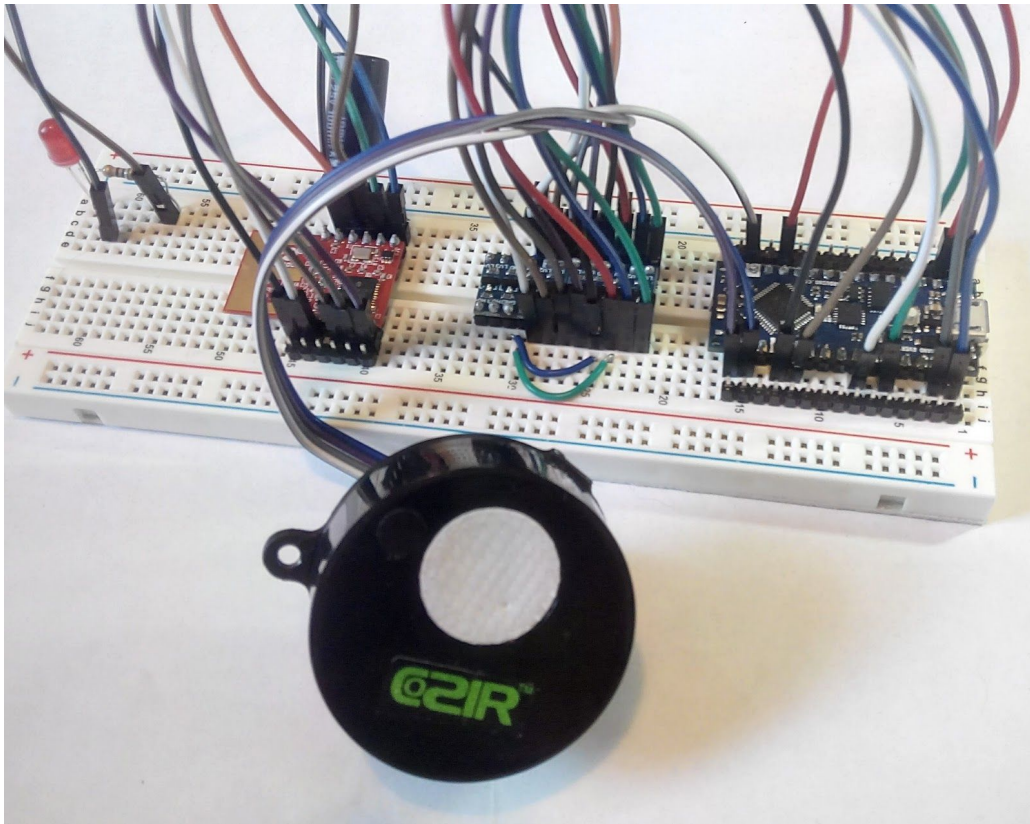


Fig V.D.2.b Sensor Node (Arduino Nano Every) Breadboard Prototype.

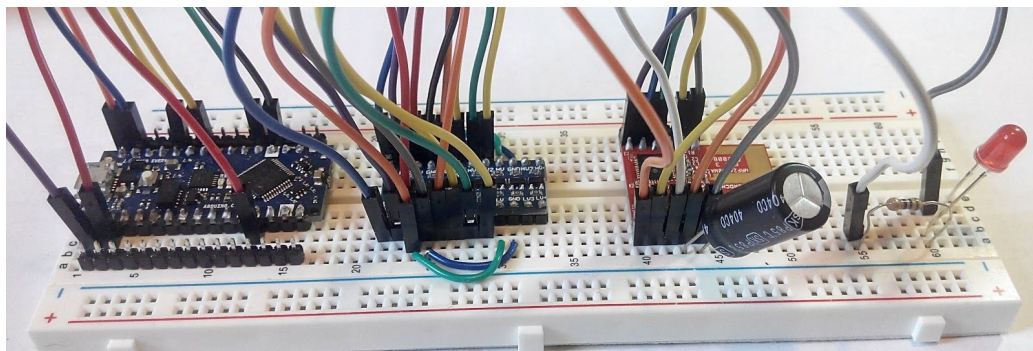


Fig V.D.2c Gateway Coordinator/Relay (Arduino Nano Every) Breadboard Prototype.

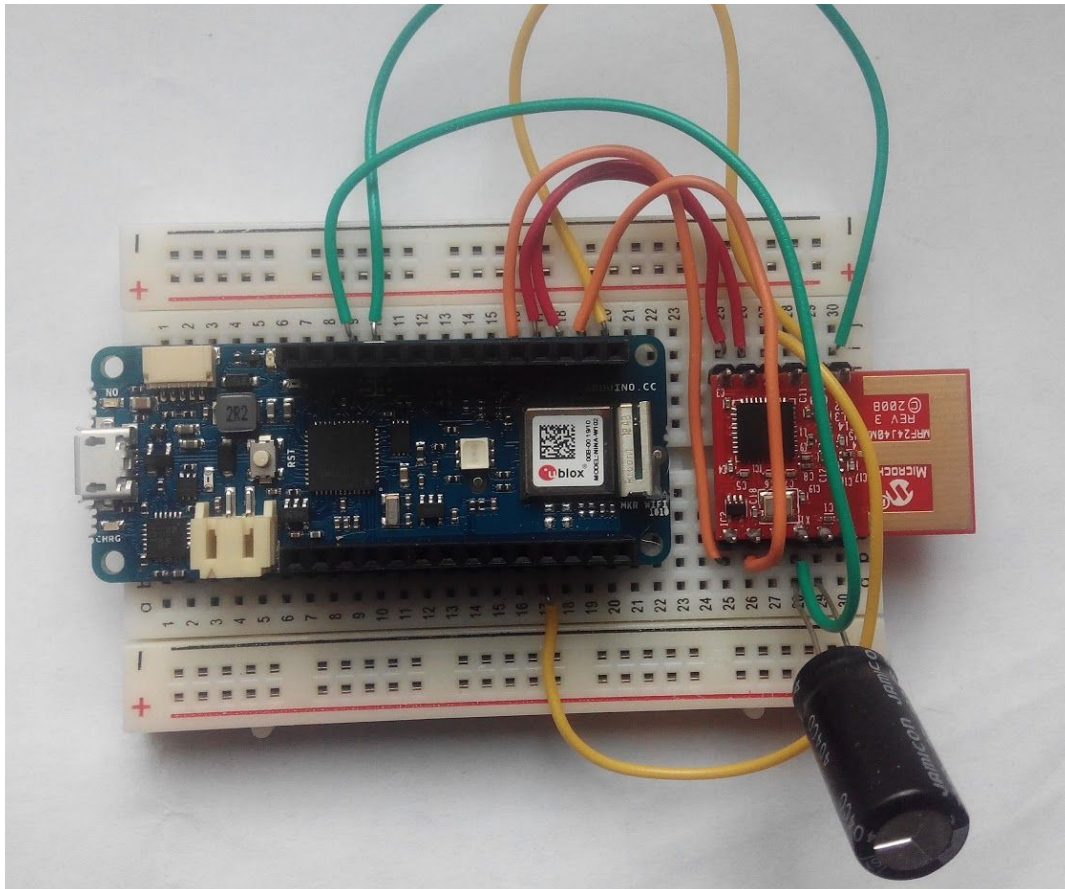


Fig V.D.2.d Gateway Coordinator (Arduino MKR WiFi 1010) Breadboard Prototype.

Debugging Transceiver Communication Issues - The process of debugging the telecommunication happened in two steps. First the focus was on the hardware: checking the connections were right, checking the voltage the RF modules were operating at was correct and finally reducing Radio Frequency Interferences (RFI). Once the pin connections were checked, voltage levels reveal inconsistencies as shown in Fig V.D.2.e. The general operating voltage of the module is 3.3V with the Arduino sending over 4V signals which would have for effect to overload, destabilise and potentially damage the transceiver (Note: this is not an issue with the MKR WiFi 1010. The I/O pins have a 3.3V max voltage rating). Therefore it was necessary to implement a level converter to ensure adequate voltage level (as described in “V.A.5 Bi-Directional Logic Level Converter”). Following the implementation measurements return balanced voltage (Shown in Fig V.D.2.f). Although, the sensor node and the gateway were still not communicating.

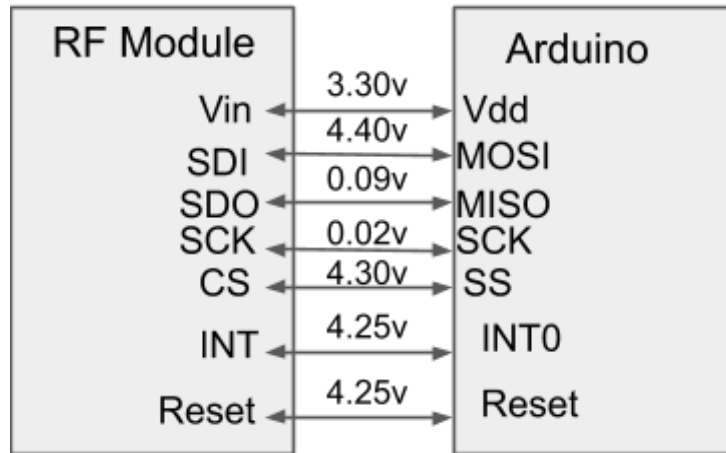


Fig V.D.2.e Voltage Level Arduino-MRF24J40MA (without bi-directional logic level converter).

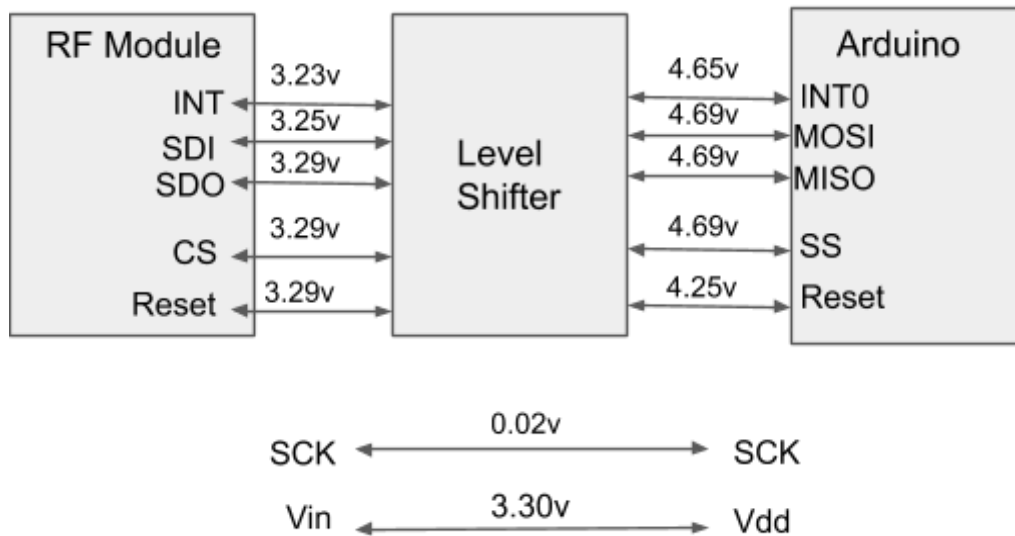


Fig V.D.2.f Voltage Level Arduino-MRF24J40MA (with bi-directional logic level converter).

The prototype being created using a breadboard with long jumper cables and low signals, the risk for resonance in the circuit and noise in the power rail are likely. The likeliness of signals interfering with the RF transceiver is high.(Horowitz, Hill and Oxford University Press, 2018). Although not available in this report, the power line was checked for noise using an oscilloscope and showing a high level of fluctuation in the 3.3V power line. On the sensor node platform, this line is shared with the CO₂ sensor which could explain the fluctuation.

Says (2017) recommends using a decoupler capacitor between the 3.3V and ground pins of the integrated circuit to limit the impact of the noise in the power source. This capacitor acts as a buffer by absorbing voltage if it is too high or releasing voltage if it is too low to essentially provide a steady level. Here again, the communication problem was not solved. Later when compared, capacitor, the transceivers dropped less incoming packets with the decoupling capacitor installed.

The next step was to focus on the software aspect of application. This resulted in the high level of detail given sections V.A.2 and V.C.1 to describe the operation of the MRF24J40MA. After checking the basic configuration of the various control registers no error was found when compared with the datasheet. The datasheet explains three reception modes are available (as explained in V.A.2 Reception). When the RF transceivers are set to promiscuous (i.e: receive all packets with a good CRC), the sensor node can transmit its measured values to the gateway (demonstrated in Fig V.D.2.g).

COM16	COM14
<pre> 23:50:29.448 -> <-----> 23:50:30.403 -> broadcasting... 23:50:30.438 -> Co2 = 23:50:30.438 -> 1048 ppm 23:50:30.438 -> <-----> 23:50:31.401 -> broadcasting... 23:50:31.435 -> Co2 = 23:50:31.435 -> 1064 ppm 23:50:31.435 -> <-----> 23:50:32.398 -> broadcasting... 23:50:32.432 -> Co2 = 23:50:32.432 -> 1077 ppm 23:50:32.467 -> <-----> 23:50:33.411 -> broadcasting... </pre>	<pre> 23:50:31.575 -> ASCII data (relevant data): 23:50:31.609 -> 1048 23:50:31.644 -> LQI/RSSI=122/254 23:50:31.644 -> Listening 23:50:32.602 -> received a packet 15 bytes long 23:50:32.637 -> 23:50:32.637 -> ASCII data (relevant data): 23:50:32.637 -> 1064 23:50:32.671 -> LQI/RSSI=121/254 23:50:32.671 -> Listening 23:50:33.619 -> received a packet 15 bytes long 23:50:33.654 -> 23:50:33.654 -> ASCII data (relevant data): 23:50:33.689 -> 1077 </pre>
<input type="checkbox"/> Autoscroll <input checked="" type="checkbox"/> Show timestamp	<input type="checkbox"/> Autoscroll <input checked="" type="checkbox"/> Show timestamp

Fig V.D.2.e COM16 Sensor Node Serial Monitor/COM14 Gateway Serial Monitor.

3. Arduino MKR WiFi 1010 Connecting to the Cloud.

The experiments presented here use the Arduino MKR WiFi 1010 and the MKR ENV Shied (shown in Fig.V.D.3.a). The purpose was to demonstrate how sensor data can be transmitted to a cloud dashboard for display. The aim is to then incorporate the MRF24J40MA Transceiver to the Arduino MRK WiFi and retransmit the data to a cloud location for processing.

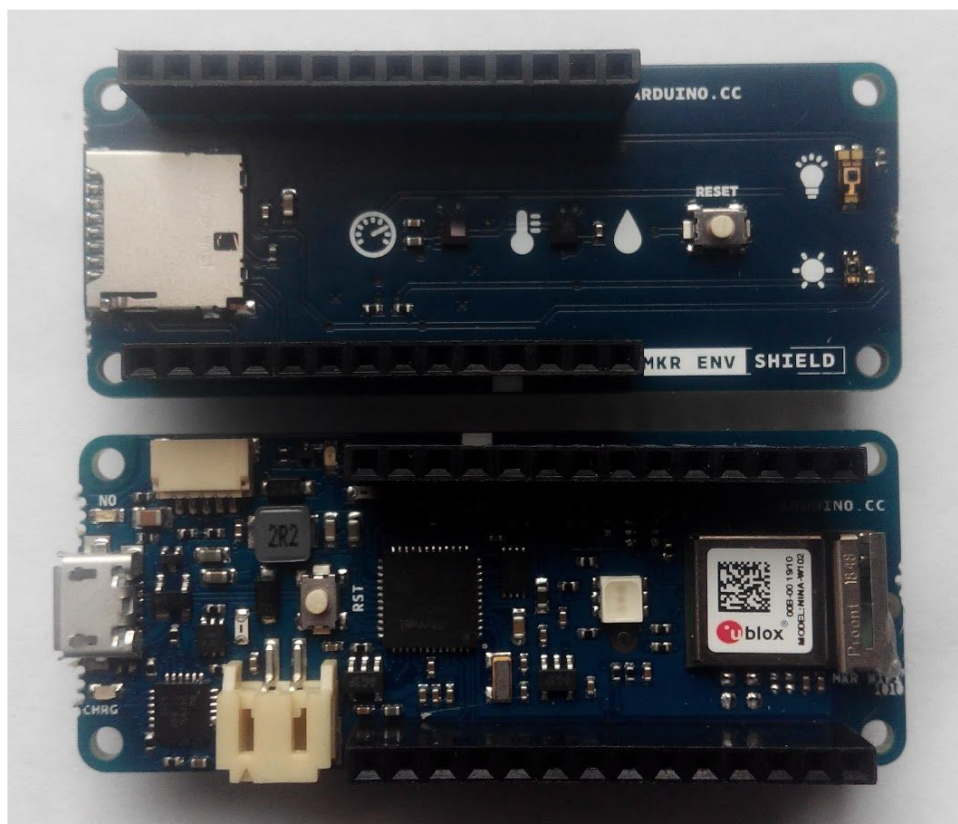
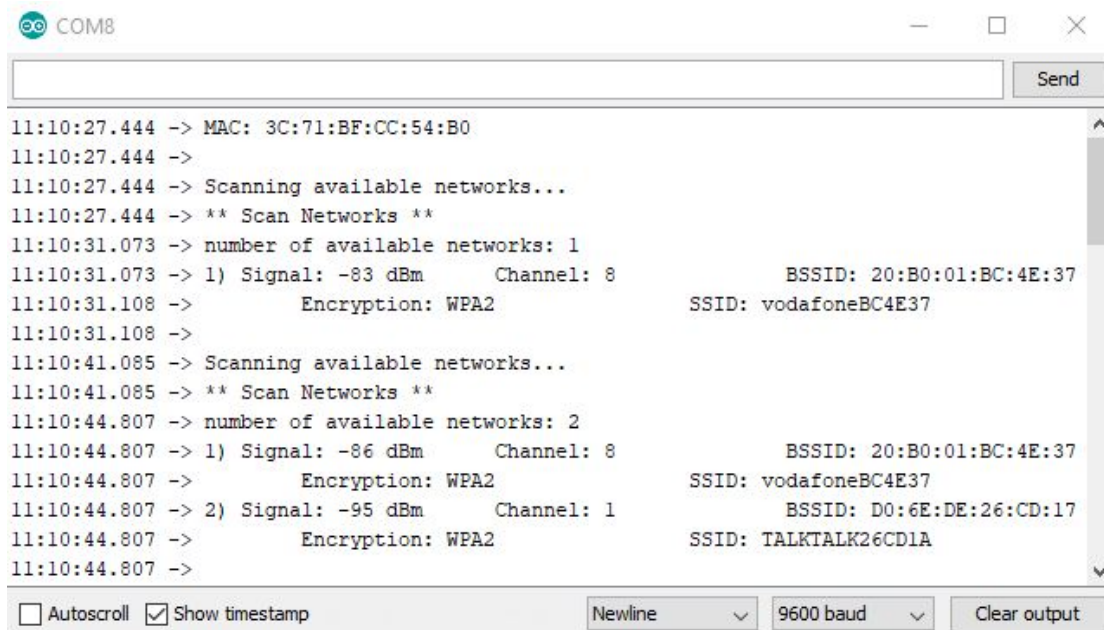


Fig.V.D.3.a MKR ENV Shield (above) and Arduino MKR WiFi 1010 (Below)

Scanning networks with the Arduino MKR1010 WiFi - The purpose of this experiment was to test the functionalities of the Arduino board. As can be noticed in Fig V.D.3.b below, the boards can scan wifi networks available in the proximity and returns various information: The board MAC Address, Service Set Identifier (SSID) and Encryption type (especially important for allowing the board to connect to a network).

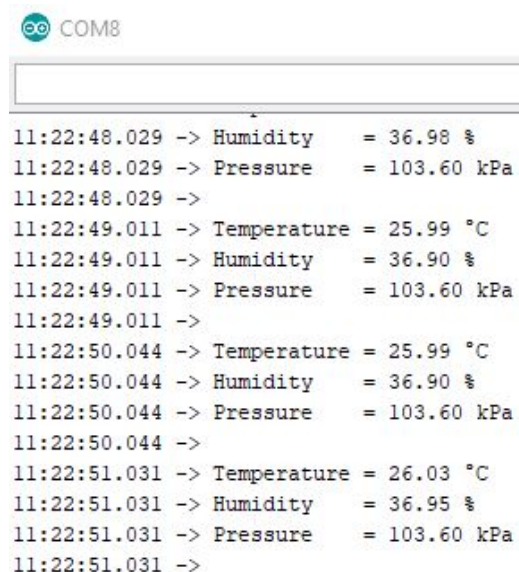


```
11:10:27.444 -> MAC: 3C:71:BF:CC:54:B0
11:10:27.444 ->
11:10:27.444 -> Scanning available networks...
11:10:27.444 -> ** Scan Networks **
11:10:31.073 -> number of available networks: 1
11:10:31.073 -> 1) Signal: -83 dBm      Channel: 8      BSSID: 20:B0:01:BC:4E:37
11:10:31.108 ->      Encryption: WPA2      SSID: vodafoneBC4E37
11:10:31.108 ->
11:10:41.085 -> Scanning available networks...
11:10:41.085 -> ** Scan Networks **
11:10:44.807 -> number of available networks: 2
11:10:44.807 -> 1) Signal: -86 dBm      Channel: 8      BSSID: 20:B0:01:BC:4E:37
11:10:44.807 ->      Encryption: WPA2      SSID: vodafoneBC4E37
11:10:44.807 -> 2) Signal: -95 dBm      Channel: 1      BSSID: D0:6E:DE:26:CD:17
11:10:44.807 ->      Encryption: WPA2      SSID: TALKTALK26CD1A
11:10:44.807 ->
```

☐ Autoscroll ☒ Show timestamp Newline 9600 baud Clear output

Fig V.D.3.b Network Scanning - Console Output.

Reading values from environmental sensors - The aim here was to test the Arduino MKR ENV Shield to then be used as a wireless sensor (demonstrated below). This board has an array of environmental sensors (temperature, humidity and pressure were selected for the example). Fig V.D.3.c shows the readings displaying on the console output.

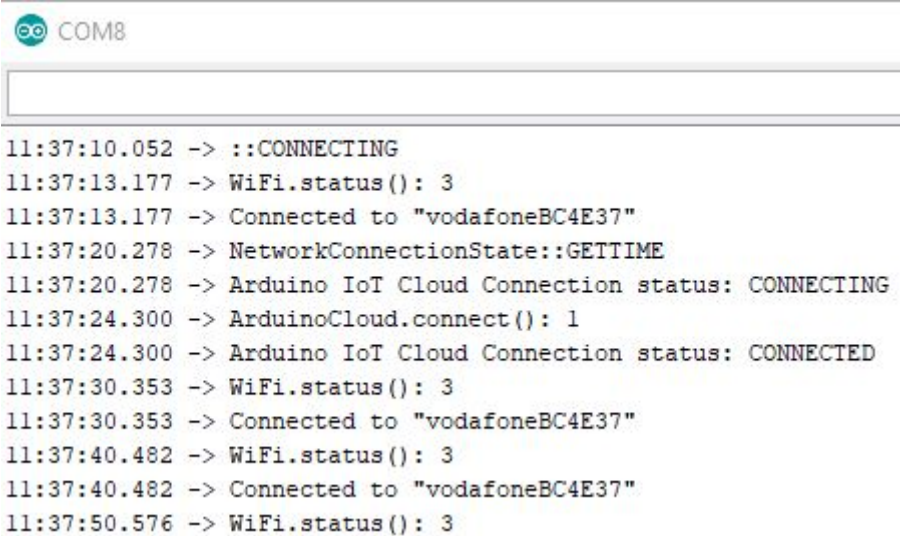


```
11:22:48.029 -> Humidity    = 36.98 %
11:22:48.029 -> Pressure    = 103.60 kPa
11:22:48.029 ->
11:22:49.011 -> Temperature = 25.99 °C
11:22:49.011 -> Humidity    = 36.90 %
11:22:49.011 -> Pressure    = 103.60 kPa
11:22:49.011 ->
11:22:50.044 -> Temperature = 25.99 °C
11:22:50.044 -> Humidity    = 36.90 %
11:22:50.044 -> Pressure    = 103.60 kPa
11:22:50.044 ->
11:22:51.031 -> Temperature = 26.03 °C
11:22:51.031 -> Humidity    = 36.95 %
11:22:51.031 -> Pressure    = 103.60 kPa
11:22:51.031 ->
```

Newline

Fig V.D.3.c Serial Monitor - Sensor Readings.

Transmit environmental data to Arduino IoT Cloud - The aim of this experiment is to demonstrate how the readings from the MRK ENV shield can be transmitted directly to the Arduino IoT Cloud to be displayed on the dashboard. The cloud is configured to recognise and transmit safely from the arduino board. The “wireless sensor” was able to connect to the cloud as demonstrated in Fig.V.D.3.d and display the value readings on the dashboard as in Fig V.D.3.e.



```
COM8
11:37:10.052 -> ::CONNECTING
11:37:13.177 -> WiFi.status(): 3
11:37:13.177 -> Connected to "vodafoneBC4E37"
11:37:20.278 -> NetworkConnectionState::GETTIME
11:37:20.278 -> Arduino IoT Cloud Connection status: CONNECTING
11:37:24.300 -> ArduinoCloud.connect(): 1
11:37:24.300 -> Arduino IoT Cloud Connection status: CONNECTED
11:37:30.353 -> WiFi.status(): 3
11:37:30.353 -> Connected to "vodafoneBC4E37"
11:37:40.482 -> WiFi.status(): 3
11:37:40.482 -> Connected to "vodafoneBC4E37"
11:37:50.576 -> WiFi.status(): 3
```

Fig V.D.3.d Serial Monitor - Connecting to Arduino Cloud.

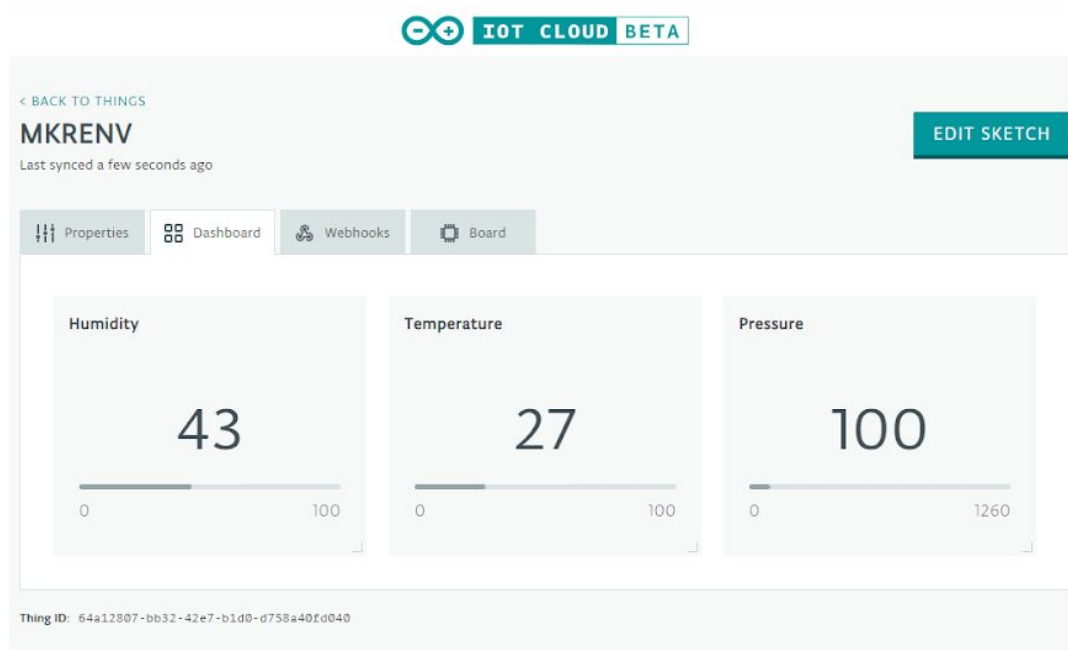


Fig V.D.3.e Arduino IoT Cloud Dashboard - Sensor Readings Display.

V. Results and Prototype Evaluation.

Results - The impact of carbon dioxide on climate change is well known. This report highlights the impact of the unbalance of the carbon cycle on environmental and human health. Although carbon capture storage technologies are ready to be implemented to absorb carbon from the atmosphere, it would be more beneficial to utilise the carbon sinking abilities of nature. By putting adequate land management practices, naturally sinking carbon would allow to restore the soil quality and reverse the effects of land degradation.

The Internet of Things being a relatively new sector evolving rapidly, the terminology can be confusing. It is important to distinguish between wireless communication, Standards and Software/Network protocol stacks to identify the best options according to the applications being developed. The different entities (Consortium and Alliance) present in the IoT market reflect that diversity with their different focuses.

The report presents the development of an IoT application known as Wireless Sensor Network with the purpose of measuring carbon dioxide. The emphasis is on the embedded system software development aspect: setup and initialisation, obtaining sensor data and communicating the data.

Discussion - The research offered in the report gives a good account as to why it would be important to monitor gases like carbon dioxide to improve environmental and public health. The IoT review gives an up to date account of the technologies available with some of their interesting aspects.

Although the description of the operation of the transceiver is fairly detailed, the final application development fell short of the objectives. Difficulties in getting a communication between two nodes to happen meant a lot of time was spent finding fault and breaking down the transmission to its basics and fault finding. The

information contained within this report would make a good starting point for someone using the MRF24J40MA for an IoT application sensor based or other.

Nonetheless, despite the difficulties, the project was a good introduction to delve into wireless communication technologies and IoT/NoT applications.

Recommendations - More investigative work is necessary to understand what other factors influence the carbon dioxide cycle and how to measure them with the aim to use multiple sensor types to derive models of the cycle within different environments.

In terms of hardware development, the power consumption evaluation provided are only theoretical values to help evaluate the battery type. A study of the different sleep modes and power saving hardware solutions with a test bench is necessary to put in place the most efficient power solution and battery to ensure the longest life possible without needing maintenance.

On the software aspect, time would be required to understand why the modules do not read messages in the normal mode of reception. This could be due to an issue with the packet frames.

Migrating the communication technology towards a cellular IoT (NB-IoT), at least for the gateway, is recommended because of the environment of applications (outdoor, in and outside of cities).

In terms of network protocols, although ZigBee is very attractive because of its software stacks, MQTT-SN seems to be the simplest to implement for such a project. It is not only very light and adaptable, it can easily be integrated in networks with an array of different communication technology.

Finally, the sensor would require extensive testing. First in a controlled environment to ensure the sensor is tuned (returning accurate values) and secondly in situ to test the network over a period of time.

References.

ACT Acorn (2018). *Act Acorn*. [online] ACORN | A scalable full-chain industrial CCS project. Available at: <https://www.actacorn.eu/about-act-acorn/act-acorn> [Accessed 1 Apr. 2020].

Allahham, A.A. and Rahman, M.A. (2018). A SMART MONITORING SYSTEM FOR CAMPUS USING ZIGBEE WIRELESS SENSOR NETWORKS. *International Journal of Software Engineering and Computer Systems*, 4(1), pp.1–14.

Arduino (2019a). *Arduino - SoftwareSerial*. [online] Arduino.cc. Available at: <https://www.arduino.cc/en/Reference/softwareSerial>.

Arduino (2019b). *Arduino - SPI*. [online] www.arduino.cc. Available at: <https://www.arduino.cc/en/reference/SPI> [Accessed 22 Mar. 2020].

Arduino (2019c). *Arduino MKR WiFi 1010 | Arduino Official Store*. [online] store.arduino.cc. Available at: <https://store.arduino.cc/arduino-mkr-wifi-1010> [Accessed 3 Apr. 2020].

Arduino (2019d). *Arduino Nano Every Specifications*. [online] store.arduino.cc. Available at: <https://store.arduino.cc/arduino-nano-every/> [Accessed 3 Apr. 2020].

Arduino (2019e). *Arduino Reference*. [online] www.arduino.cc. Available at: <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/> [Accessed 25 Mar. 2020].

Arduino (2020). *Ground Power LED Internal Pin SWD Pin Digital Pin Analog Pin Other Pin Microcontroller's Port Default MOSI (SC5) (SC5) (SC2) (SC2) (SC1) (SC1)*. [online] Available at: https://content.arduino.cc/assets/Pinout-MKRWifi1010_latest.pdf [Accessed 3 Apr. 2020].

BHF (2020). *Heart and circulatory deaths related to air pollution could exceed 160,000 by 2030*. [online] British Heart Foundation. Available at: <https://www.bhf.org.uk/what-we-do/news-from-the-bhf/news-archive/2020/january/heart-and-circulatory-deaths-related-to-air-pollution-could-exceed-160000-over-next-decade> [Accessed 1 Apr. 2020].

L. Butgereit and A. Nickless, "Capturing, Calculating, and Disseminating Real-Time CO₂ Emissions and CO₂ Flux Measurements via Twitter in a Smart City," *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, Beijing, 2013, pp. 2013-2017.

Carbon Capture and Storage Association (2011). *What is CCS? – The Carbon Capture & Storage Association (CCSA)*. [online] Ccsassociation.org. Available at: <http://www.ccsassociation.org/what-is-ccs/> [Accessed 1 Apr. 2020].

Carrington, D., Smears, L., Blight, G., Roberts, S. and Hulley-Jones, F. (2018). *Revealed: air pollution may be damaging “every organ in the body.”* [online] the Guardian. Available at: <https://www.theguardian.com/environment/ng-interactive/2019/may/17/air-pollution-may-be-damaging-every-organ-and-cell-in-the-body-finds-global-review>.

Carrington, D. and Taylor, M. (2018). *Air pollution is the ‘new tobacco’, warns WHO head*. [online] the Guardian. Available at: <https://www.theguardian.com/environment/2018/oct/27/air-pollution-is-the-new-tobacco-warns-who-head> [Accessed 26 Sep. 2019].

Choudhuri, R. (2017). *Learn Arduino prototyping in 10 days : build it, test it, learn, try again!* Birmingham: Packt Pub.

cplusplus.com (2000). *itoa - C++ Reference*. [online] www.cplusplus.com. Available at: <http://www.cplusplus.com/reference/cstdlib/itoa/> [Accessed 25 Mar. 2020].

Davis, N. (2017). 65% of British public support new Clean Air Act, says survey. *The Guardian*. [online] 14 Feb. Available at: <https://www.theguardian.com/cities/2017/feb/14/65-percent-british-public-want-clean-air-act-pollution-harmful-uk-survey> [Accessed 30 Mar. 2020].

Devadiga, K. (2011). *2011S-iot - ESG Seminar - Aalto University Wiki*. [online] wiki.aalto.fi. Available at: <https://wiki.aalto.fi/display/esgsem/2011S-iot?preview=/59704179/63767691/devadiga-802-15-4-and-the-iot.pdf> [Accessed 3 Apr. 2020].

Dhingra, S., Madda, R.B., Gandomi, A.H., Patan, R. and Daneshmand, M. (2019). Internet of Things Mobile - Air Pollution Monitoring System (IoT-Mobair). *IEEE Internet of Things Journal*, 6(3), pp.1–1.

ejshea (2019). *Connecting an N-Channel MOSFET*. [online] Arduino Project Hub. Available at: <https://create.arduino.cc/projecthub/ejshea/connecting-an-n-channel-mosfet-7e0242> [Accessed 29 Mar. 2020].

Elliot Williamson (2014). *Make : AVR programming*. Sebastopol: Maker Media. C.

European Environment Agency (2016). 8. *Climate change*. [online] European Environment Agency. Available at: <https://www.eea.europa.eu/publications/92-9167-205-X/page009.html> [Accessed 1 Apr. 2020].

European Environment Agency (2020). *carbon sink*. [online] European Environment Agency. Available at: <https://www.eea.europa.eu/help/glossary/eea-glossary/carbon-sink> [Accessed 1 Apr. 2020].

Food And Agriculture Organization Of The United Nations (2004). *Carbon sequestration in dryland soils*. Rome: Food And Agriculture Organization For The United Nations.

Food And Agriculture Organization Of The United Nations (2020). *Soil Carbon Sequestration | FAO SOILS PORTAL | Food and Agriculture Organization of the United Nations*. [online] Fao.org. Available at: <http://www.fao.org/soils-portal/soil-management/soil-carbon-sequestration/en/> [Accessed 6 Jan. 2020].

Foot, K.D. (2016). *A Brief History of the Internet of Things - DATAVERSITY*. [online] DATAVERSITY. Available at: <https://www.dataversity.net/brief-history-internet-things/> [Accessed 30 Mar. 2020].

Franzluebbers, A.J. and Doraiswamy, P.C. (2007). Carbon Sequestration and Land Degradation. *Climate and Land Degradation*, pp.343–358.

GSS (2018). *GSS Sensor User Guide*. [online] Available at: <https://www.gassensing.co.uk/wp-content/uploads/2018/11/GSS-Sensor-User-Guide.pdf> [Accessed 3 Apr. 2020].

G. S. Gupta and V. M. Quan, "Multi-sensor integrated system for wireless monitoring of greenhouse environment," *2018 IEEE Sensors Applications Symposium (SAS)*, Seoul, 2018, pp. 1-6.

Horowitz, P., Hill, W. and Oxford University Press (2018). *The art of electronics*. New York: Cambridge University Press.

<http://www.injsoft.se>, I.A. (2005). *ASCII Code - The extended ASCII table*. [online] www.ascii-code.com. Available at: <https://www.ascii-code.com/> [Accessed 25 Mar. 2020].

Hughes, J.M. (2016). *Arduino : a technical reference a handbook for technicians, engineers, and makers*. Sebastopol, Ca: O'reilly Media.

Hunkeler, U., Truong, H. and Stanford-Clark, A. (2019). *MQTT-S -A Publish/Subscribe Protocol For Wireless Sensor Networks*. [online] Available at: <https://sites.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/mqtt-s.pdf> [Accessed 3 Apr. 2020].

Hwang, Y. (2020). *Cellular IoT Explained - NB-IoT vs. LTE-M vs. 5G and More*. [online] IoT For All. Available at: <https://www.iotforall.com/cellular-iot-explained-nb-iot-vs-lte-m/> [Accessed 2 Apr. 2020].

Institute for Materials and Processes (n.d.). *ACCA: Atmospheric Carbon Capture | Carbon Capture*. [online] www.carboncapture.eng.ed.ac.uk. Available at: <https://www.carboncapture.eng.ed.ac.uk/acca-atmospheric-carbon-capture> [Accessed 1 Apr. 2020].

Institute Of Electrical And Electronics Engineers (2003). *Part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANS)*. New York: Institute of Electrical And Electronics Engineers.

IUCN (2015). *Land degradation and climate change*. [online] IUCN. Available at: <https://www.iucn.org/resources/issues-briefs/land-degradation-and-climate-change> [Accessed 1 Apr. 2020].

J. Voas, B. Agresti and Laplante, P.A. (2018). A Closer Look at IoT 's Things. *IT Professional*, 20(3), pp.11–14.

Langbridge, J.A. (2015). *Arduino sketches : tools and techniques for programming wizardry*. Hoboken: John Wiley & Sons.

Margolis, M. (2018). *Arduino Cookbook*. O'reilly Media, Incorporated.

MET (2019). *Mauna Loa carbon dioxide forecast for 2020*. [online] Met Office. Available at: <https://www.metoffice.gov.uk/research/climate/seasonal-to-decadal/long-range/forecasts/co2-forecast>.

Microchip (2008a). *MRF24J40MA Data Sheet 2.4 GHz IEEE Std. 802.15.4™ RF Transceiver Module*. [online] Available at: <http://ww1.microchip.com/downloads/en/DeviceDoc/MRF24J40MA-Data-Sheet-70000329C.pdf> [Accessed 3 Apr. 2020].

Microchip (2008b). *Preliminary MRF24J40 Data Sheet IEEE 802.15.4™ 2.4 GHz RF Transceiver*. [online] Available at: <https://ww1.microchip.com/downloads/en/DeviceDoc/39776C.pdf> [Accessed 3 Apr. 2020].

Microchip (2018). *ATmega809/1609/3209/4809 - 48-pin 48-pin Data Sheet -megaAVR® 0-series*. [online] Available at: https://content.arduino.cc/assets/Nano-Every_processor-48-pin-Data-Sheet-megaAVR-0-series-DS40002016B.pdf [Accessed 3 Apr. 2020].

Microchip (2019). *MiWi™ MiWi™ Software Design Guide*. [online] Available at: <http://ww1.microchip.com/downloads/en/DeviceDoc/MiWi-Software-Design-Guide-User-Guide-DS50002851B.pdf> [Accessed 3 Apr. 2020].

Ontl, T.A. and Schulte, L.. (2012). *Soil Carbon Storage | Learn Science at Scitable*. [online] Nature.com. Available at: <https://www.nature.com/scitable/knowledge/library/soil-carbon-storage-84223790/>.

National Institute of Standards and Technology (2016). *NIST Special Publication 800-183 Networks of 'Things'*. Gaithersburg: National Institute of Standards and Technology.

PA Media, P. (2020). *Air pollution could kill 160,000 in next decade – report*. [online] the Guardian. Available at: <https://www.theguardian.com/environment/2020/jan/13/air-pollution-kill-160000-next-decade-report>.

Palsson, K. (2019). *MRF24J40 Arduino Library*. [online] GitHub. Available at: <https://github.com/karlp/Mrf24j40-arduino-library> [Accessed 22 Mar. 2020].

Priestley International Centre for Climate (2018). *"Critical" levels of land degradation intensifying climate change : Priestley International Centre for Climate*. [online] climate.leeds.ac.uk. Available at: <https://climate.leeds.ac.uk/news/critical-levels-of-land-degradation-intensifying-climate-change/> [Accessed 1 Apr. 2020].

Rand, P. (2020). *What is Cellular IoT?* [online] blog.nordicsemi.com. Available at: <https://blog.nordicsemi.com/getconnected/what-is-cellular-iot> [Accessed 2 Apr. 2020].

Says, G. (2017). *What Are Decoupling Capacitors in 5 Minutes | EAGLE | Blog*. [online] Eagle Blog. Available at: <https://www.autodesk.com/products/eagle/blog/what-are-decoupling-capacitors/> [Accessed 29 Mar. 2020].

SCCS (2019). *ACT Acorn collaboration shows how UK's assets can accelerate Europe's climate action*. [online] www.sccs.org.uk. Available at: <https://www.sccs.org.uk/news-events/recent-news/507-act-acorn-collaboration-shows-how-uks-assets-can-accelerate-europes-climate-action> [Accessed 1 Apr. 2020].

Scherz, P., Monk, S. and McGraw-Hill Education (2016). *Practical electronics for inventors*. New York Etc.: McGraw Hill Education, Cop.

See, C., Horoshenkov, K., abd-alhmeed, R., Hu, Y. and Tait, S. (2011). A Low Power Wireless Sensor Network for Gully Pot Monitoring in Urban Catchments. *IEEE Sensors Journal*, 12(5).

Silicon Labs (2013). *The Evolution of Wireless Sensor Networks Origin and History of Wireless Sensor Networks*. [online] Available at:

<https://www.silabs.com/documents/public/white-papers/evolution-of-wireless-sensor-networks.pdf>
[Accessed 3 Apr. 2020].

SparkFun Electronics (2020). *Reducing Arduino Power Consumption - learn.sparkfun.com*. [online] learn.sparkfun.com. Available at:
<https://learn.sparkfun.com/tutorials/reducing-arduino-power-consumption#introduction> [Accessed 20 Mar. 2020].

SST (2016). *O 2 SENSORS -LuminOx User's Guide*. [online] Available at:
https://14core.com/wp-content/uploads/2017/10/LuminOx-UserGuide_rev1.pdf [Accessed 3 Apr. 2020].

Stanford-Clark, A. and Truong, H. (2013). *MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2*. [online] Available at:
https://www.mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf [Accessed 3 Apr. 2020].

Tillaart, R. and Hawthorne, M. (2018). *CoZIR Arduino Library*. [online] GitHub. Available at:
<https://github.com/RobTillaart/Arduino/tree/master/libraries/Cozir> [Accessed 22 Mar. 2020].

Timur Mirzoev (2014). Low rate wireless personal area networks (LR-WPAN 802.15.4 standard). *CoRR*, [online] abs/1404.2345. Available at: <http://arxiv.org/abs/1404.2345>.

trsridhar, arvindpdmn (2018). *IoT Alliances and Consortiums*. [online] Devopedia. Available at:
<https://devopedia.org/iot-alliances-and-consortiums> [Accessed 28 Mar. 2020].

u-blox (2015). *Narrowband IoT (NB-IoT)*. [online] u-blox. Available at:
<https://www.u-blox.com/en/narrowband-iot-nb-iot> [Accessed 2 Apr. 2020].

u-blox (2018a). *LTE-M*. [online] u-blox. Available at: <https://www.u-blox.com/en/technologies/lte-cat-m>
[Accessed 2 Apr. 2020].

u-blox (2018b). *Narrowband IoT (NB-IoT)*. [online] u-blox. Available at:
<https://www.u-blox.com/en/technologies/narrowband-iot-nb-iot> [Accessed 2 Apr. 2020].

UKRI (2019). *Strategic Priorities Fund: Clean Air Programme*. [online] Available at:
<https://nerc.ukri.org/research/funded/programmes/clean-air/news/ao-networks/ao-clean-air-networks-call/> [Accessed 3 Apr. 2020].

Voas, J. (2016). Demystifying the Internet of Things. *Computer*, 49(6), pp.80–83.

Wang, C., Jiang, T. and Zhang, Q. (2019). *ZigBee® network protocols and applications*. Boca Raton: Auerbach.

WHO (2012). WHO | Land degradation and desertification. *Who.int*. [online] Available at: <https://www.who.int/globalchange/ecosystems/desert/en/>.

ZigBee Alliance (2016a). *JupiterMesh*. [online] Zigbee Alliance. Available at: <https://zigbeealliance.org/solution/jupitermesh/> [Accessed 3 Apr. 2020].

ZigBee Alliance (2016b). *Smart Energy*. [online] Zigbee Alliance. Available at: <https://zigbeealliance.org/solution/smart-energy/> [Accessed 3 Apr. 2020].

ZigBee Alliance (2017a). *Dotdot*. [online] Zigbee Alliance. Available at: <https://zigbeealliance.org/solution/dotdot/> [Accessed 3 Apr. 2020].

ZigBee Alliance (2017b). *Green Power*. [online] Zigbee Alliance. Available at: <https://zigbeealliance.org/solution/green-power/> [Accessed 3 Apr. 2020].

ZigBee Alliance (2019). *Zigbee - Zigbee Alliance*. [online] Zigbee Alliance. Available at: <https://zigbeealliance.org/solution/zigbee/> [Accessed 3 Apr. 2020].

Appendices.

Appendix A - Sensor Node Applications Code.

```
//Sensor Libraries and Variables Declaration.
#include <SoftwareSerial.h> //Serial port library.
#include "cozir.h" //CO2 sensor library.
SoftwareSerial nss(0,1); //Rx Tx pin assignment.
COZIR czr(nss); //Pass serial pins to the CoZIR library.
String val= ""; //Holds the string of the value.
int co2 =0; //Holds the actual CO2 value.
double multiplier = 1; //each range of sensor has a different value.
uint8_t buffer[25]; //buffer to read data from serial port.
uint8_t ind =0; //index variable to iterate over buffer.

//MRF24J40MA Libraries, Pin Assignment and Variables Declaration.
#include <SPI.h> //SPI port library.
#include "mrf24j.h" //MRF24J40MA Library.
const int pin_reset = 6; //Reset pin assignment.
const int pin_cs = 8; //CS pin assignment.
const int pin_interrupt = 2; //Interrupt pin assignment.
const int pin_sclock = 13; //Clock output pin assignment.
Mrf24j mrf(pin_reset, pin_cs, pin_interrupt, pin_sclock); //Pass pins to MRF24J
library.
char msg_send[4]; //string storing the data to be sent
long last_time; //
long tx_interval = 1000;

void setup() {
//Sensor Setup.
Serial.begin(9600); //Start Serial for programme status display.
Serial1.begin(9600); //Start Serial connection with Sensor.
delay(3000);
//czr.CalibrateFreshAir(); //Uncomment to use autocalibration.
czr.SetOperatingMode(CZR_POLLING); //Set sensor to polling mode.
//czr.SetOperatingMode(CZR_STREAMING); //Uncomment to use streaming mode.

//RF Module setup.
mrf.reset(); //Perform a hardware reset.
mrf.init(); //Initialisation routine of the transceiver.
mrf.set_promiscuous(true); //Receive any packet on this channel
//Network Setup:
mrf.set_pan(0xcafe); //Network ID
mrf.address16_write(0x6001); //Node Address
//Interrupts Setup.
attachInterrupt(digitalPinToInterrupt(pin_interrupt), interrupt_routine, LOW);
```

```

//interrupt 0 equivalent to pin 2(INT0).
  last_time = millis(); //Time to set application start time (post setup).
}
//Interrupt Routine for RF Module.
void interrupt_routine() {
  mrf.interrupt_handler(); //mrf24 interrupt routine from library.
}

void loop() {
  //-----//
  //Get CO2 reading and package to a variable msg_send.
  //-----//
  //Read incoming bytes into a buffer until we get '0x0A'(ASCII value for
  new-line).
  Serial1.println("Z\r\n"); //Send request command for CO2.
  while(buffer[ind-1] != 0x0A) //While previous character not "new-line".
  {
    if(Serial1.available()) //check if serial is ready.
    {
      buffer[ind] = Serial1.read(); //Characters from serial to buffer.
      ind++; // increase index for each character.
    }
  }
  unpack(); //Once character = '0x0A', unpack the buffer data.
  //-----//
  //Package and Send Message with RF Module.
  //-----//
  mrf.check_flags(&rx_handle, &tx_handle); //Check flags to launch
  appropriate routine.
  unsigned long current_time = millis(); // Get current time.
  if (current_time - last_time > tx_interval) { //Constrain transmission to
  once every second.
    Serial.println("broadcasting..."); //Print broadcasting state.
    mrf.send16(0x4202, msg_send); //Send string to defined address .
  }
  //Message is sent, reset the buffer values.
  ind=0; //Reset the buffer index.
  val=""; //Reset the value string.
}
void unpack()
{
  //-----//
  //Unpack the buffer to get the CO2 reading value.
  //-----//
  for(int i=0; i < ind+1; i++) //Unpacking loop iteration.
  {
    if(buffer[i] == 'z') //once we hit the 'z' character stops the unpacking
    loop.
    break;
  }
}

```

```

        if((buffer[i] != 0x5A)&&(buffer[i] != 0x20)) //Ignore 'Z' and spaces.
        {
            val += buffer[i]-48; //Subtract 48 from each numerical character to get to
the actual numerical value.
        }
    }

    co2 = (multiplier * val.toInt()); //val.toInt()now we multiply the value by a
factor specific to the sensor as per Software Guide.
    //Print the CO2 value to serial monitor.
    Serial.println( "Co2 = ");
    Serial.print(co2);
    Serial.println(" ppm");
    Serial.println("<----->");
    delay(1000);
    itoa(co2, msg_send, DEC); //Convert to string array msg_send.
    //Serial.println(msg_send); // Uncomment to print the value to be sent.
}
//-----//
//Reception of packet management routine for RF Module.
//-----//
void rx_handle() { //Print data from TX FIFO stored in buffers.
    Serial.print("received a packet
");Serial.print(mrf.get_rxinfo()->frame_length, DEC);Serial.println(" bytes
long");

    if(mrf.get_bufferPHY()){
        Serial.println("Packet data (PHY Payload:"); //Print all characters in
the data.
        for (int i = 0; i < mrf.get_rxinfo()->frame_length; i++) { //iterate over
each character in the buffer until the end of the frame.
            Serial.print(mrf.get_rxbuf()[i]);
        }
    }

    Serial.println("\r\nASCII data (relevant data):"); //Print the msg_send (CO2
value).
    for (int i = 0; i < mrf.rx_datalength(); i++) { //iterate over each character
in the buffer for the length of the data buffer.
        Serial.write(mrf.get_rxinfo()->rx_data[i]);
    }

    Serial.print("\r\nLQI/RSSI="); //Print signal related information (LQI and
RSSI).
    Serial.print(mrf.get_rxinfo()->lqi, DEC);
    Serial.print("/");
    Serial.println(mrf.get_rxinfo()->rssi, DEC);
}

void tx_handle() { //Confirmation of good transmission of packet.

```

```

    if (mrf.get_txinfo()->tx_ok) {
        Serial.println("TX went ok, got ack");
    } else {
        Serial.print("TX failed after
");Serial.print(mrf.get_txinfo()->retries);Serial.println(" retries\n");
    }
}
}

```

Appendix B - Gateway Coordinator Applications Code.

```

//MRF24J40MA Libraries, Pin Assignment and Variables Declaration.
#include <SPI.h> //SPI port library.
#include "mrf24j.h" //MRF24J40MA Library.
const int pin_reset = 6; //Reset pin assignment.
const int pin_cs = 8; //CS pin assignment.
const int pin_interrupt = 2; //Interrupt pin assignment.
const int pin_sclock = 13; //Clock output pin assignment.
Mrf24j mrf(pin_reset, pin_cs, pin_interrupt, pin_sclock); //Pass pins to MRF24J
library.
char msg_send[4]; //string storing the data to be sent
long last_time; //
long tx_interval = 1000;

Void setup(){
//RF Module setup.
    mrf.reset(); //Perform a hardware reset.
    mrf.init(); //Initialisation routine of the transceiver.
    mrf.set_promiscuous(true); //Receive any packet on this channel
//Network Setup:
    mrf.set_pan(0xcafe); //Network ID
    mrf.address16_write(0x4202); //Node Address
//Interrupts Setup.
    attachInterrupt(digitalPinToInterrupt(pin_interrupt), interrupt_routine, LOW);
//interrupt 0 equivalent to pin 2(INT0).
    last_time = millis(); //Time to set application start time (post setup).
}
//Interrupt Routine for RF Module.
void interrupt_routine() {
    mrf.interrupt_handler(); //mrf24 interrupt routine from library.
}
//-----//
//Listen for Message with RF Module.
//-----//
    mrf.check_flags(&rx_handle, &tx_handle); //Check flags to launch
appropriate routine.
    unsigned long current_time = millis(); // Get current time.
    if (current_time - last_time > tx_interval) { //Constrain transmission to
once every second.

```

```

        Serial.println("Listening");
    }
    //-----//
    //Reception of packet management routine for RF Module.
    //-----//
    void rx_handle() { //Print data from TX FIFO stored in buffers.
        Serial.print("received a packet
");Serial.print(mrf.get_rxinfo()->frame_length, DEC);Serial.println(" bytes
long");

        if(mrf.get_bufferPHY()){
            Serial.println("Packet data (PHY Payload:"); //Print all characters in
the data.
            for (int i = 0; i < mrf.get_rxinfo()->frame_length; i++) { //iterate over
each character in the buffer until the end of the frame.
                Serial.print(mrf.get_rxbuf()[i]);
            }
        }

        Serial.println("\r\nASCII data (relevant data:"); //Print the msg_send (CO2
value).
        for (int i = 0; i < mrf.rx_datalength(); i++) { //iterate over each character
in the buffer for the length of the data buffer.
            Serial.write(mrf.get_rxinfo()->rx_data[i]);
        }

        Serial.print("\r\nLQI/RSSI="); //Print signal related information (LQI and
RSSI).
        Serial.print(mrf.get_rxinfo()->lqi, DEC);
        Serial.print("/");
        Serial.println(mrf.get_rxinfo()->rssi, DEC);
    }

    void tx_handle() { //Confirmation of good transmission of packet.
        if (mrf.get_txinfo()->tx_ok) {
            Serial.println("TX went ok, got ack");
        } else {
            Serial.print("TX failed after
");Serial.print(mrf.get_txinfo()->retries);Serial.println(" retries\n");
        }
    }
}

```

Appendix C - CoziR Sensor Library.

Header File:

```
// FILE: Cozir.h
```

```

//    AUTHOR: Rob Tillaart & Michael Hawthorne
//    VERSION: 0.1.01
//    PURPOSE: library for COZIR range of sensors for Arduino
//    URL: http://www.cozir.com/
//    DATASHEET:
http://www.co2meters.com/Documentation/Datasheets/COZIR-Data-Sheet-RevC.pdf
// USER GUIDE:
http://www.co2meters.com/Documentation/Manuals/COZIR-Software-User-Guide-AL12-RevA.pdf
// READ DATASHEET BEFORE USE OF THIS LIB !
//
// Released to the public domain
//

#ifdef Cozir_h
#define Cozir_h

#include "SoftwareSerial.h"

#if defined(ARDUINO) && ARDUINO >= 100
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif

#define COZIR_LIB_VERSION 0.1.01

// OUTPUTFIELDS
// See datasheet for details.
// These defines can be OR-ed for the SetOutputFields command
#define CZR_LIGHT            0x2000
#define CZR_HUMIDITY         0x1000
#define CZR_FILTLED          0x0800
#define CZR_RAWLED           0x0400
#define CZR_MAXLED           0x0200
#define CZR_ZEROPOINT        0x0100
#define CZR_RAWTEMP           0x0080
#define CZR_FILTTEMP          0x0040
#define CZR_FILTLEDSIGNAL    0x0020
#define CZR_RAWLEDSIGNAL     0x0010
#define CZR_SENSTEMP          0x0008
#define CZR_FILTCO2           0x0004
#define CZR_RAWCO2            0x0002
#define CZR_NONE              0x0001

// easy default setting for streaming
#define CZR_HTC                (CZR_HUMIDITY | CZR_RAWTEMP | CZR_RAWCO2)
// not in datasheet for debug only
#define CZR_ALL                 0x3FFF

```

```

// OPERATING MODES
#define CZR_COMMAND          0x00
#define CZR_STREAMING        0x01
#define CZR_POLLING          0x02

class COZIR
{
public:
    COZIR(SoftwareSerial&);//: CZR_Serial(nss)

    void SetOperatingMode(uint8_t mode);

    float Celsius();
    float Fahrenheit();
    float Humidity();
    float Light();
    uint16_t CO2();

    uint16_t FineTuneZeroPoint(uint16_t , uint16_t);
    uint16_t CalibrateFreshAir();
    uint16_t CalibrateNitrogen();
    uint16_t CalibrateKnownGas(uint16_t );
    uint16_t CalibrateManual(uint16_t );
    uint16_t SetSpanCalibrate(uint16_t );
    uint16_t GetSpanCalibrate();

    void SetDigiFilter(uint8_t );
    uint8_t GetDigiFilter();

    void SetOutputFields(uint16_t );
    void GetRecentFields();

    void SetEEPROM(uint8_t , uint8_t );
    uint8_t GetEEPROM(uint8_t );

    void GetVersionSerial();
    void GetConfiguration();

private:
    SoftwareSerial& CZR_Serial;
    char buffer[20];
    void Command(char* );
    uint16_t Request(char* );
};

#endif

```

C++ File:

```

//      FILE: Cozir.cpp
//      AUTHOR: Rob Tillaart & Michael Hawthorne
//      VERSION: 0.1.01
//      PURPOSE: library for COZIR range of sensors for Arduino
//      URL: http://www.cozir.com/
//      DATASHEET:
http://www.co2meters.com/Documentation/Datasheets/COZIR-Data-Sheet-RevC.pdf
// USER GUIDE:
http://www.co2meters.com/Documentation/Manuals/COZIR-Software-User-Guide-AL12-RevA.pdf
// READ DATASHEET BEFORE USE OF THIS LIB !
//
// Released to the public domain
//

#include "Cozir.h"
#include "SoftwareSerial.h"

////////////////////////////////////
//
// CONSTRUCTOR
//
COZIR::COZIR(SoftwareSerial& nss) : CZR_Serial(nss)
{
    CZR_Serial.begin(9600);
}

////////////////////////////////////
//
// OPERATING MODE
//
// note: use CZR_COMMAND to minimize power consumption
// CZR_POLLING and CZR_STREAMING use an equally amount
// of power as both sample continuously...
//
void COZIR::SetOperatingMode(uint8_t mode)
{
    sprintf(buffer, "K %u", mode);
    Command(buffer);
}

////////////////////////////////////
//
// POLLING MODE
//
// you need to set the polling mode explicitly before
// using these functions. SetOperatingMode(CZR_POLLING);
// this is the default behaviour of this Class but
// not of the sensor!!
//

```



```

float COZIR::Fahrenheit()
{
    return (Celsius() * 1.8) + 32;
}

float COZIR::Celsius()
{
    float f = 0;
    uint16_t rv = Request("T");
    if (rv < 1000)
    {
        f = 0.1 * rv;
    }
    else
    {
        f = -0.1 * (rv-1000);
    }
    return f;
}

float COZIR::Humidity()
{
    return 0.1 * Request("H");
}

// TODO UNITS UNKNOWN
float COZIR::Light()
{
    return 1.0 * Request("L");
}

uint16_t COZIR::CO2()
{
    return Request("Z");
}

// CALIBRATION - USE THESE WITH CARE
// use these only in pollingmode (on the Arduino)

// FineTuneZeroPoint()
// a reading of v1 will be reported as v2
// sort of mapping
// check datasheet for detailed description
uint16_t COZIR::FineTuneZeroPoint(uint16_t v1, uint16_t v2)
{
    sprintf(buffer, "F %u %u", v1, v2);
    return Request(buffer);
}

// mostly the default calibrator

```

```

uint16_t COZIR::CalibrateFreshAir()
{
    return Request("G");
}

uint16_t COZIR::CalibrateNitrogen()
{
    return Request("U");
}

uint16_t COZIR::CalibrateKnownGas(uint16_t value)
{
    sprintf(buffer, "X %u", value);
    return Request(buffer);
}

// NOT RECOMMENDED, see datasheet
uint16_t COZIR::CalibrateManual(uint16_t value)
{
    return 0;
    //sprintf(buffer, "u %u", value);
    //return Request(buffer);
}

// NOT RECOMMENDED, see datasheet
uint16_t COZIR::SetSpanCalibrate(uint16_t value)
{
    return 0;
    //sprintf(buffer, "S %u", value);
    //return Request(buffer);
}

// NOT RECOMMENDED, see datasheet
uint16_t COZIR::GetSpanCalibrate()
{
    return Request("s");
}

// DIGIFILTER, use with care
// default value = 32,
// 1=fast (noisy) 255=slow (smoothed)
// 0 = special. details see datasheet
void COZIR::SetDigiFilter(uint8_t value)
{
    sprintf(buffer, "A %u", value);
    Command(buffer);
}

uint8_t COZIR::GetDigiFilter()
{

```

```

    return Request("a");
}

////////////////////////////////////
//
// STREAMING MODE
//
// outputfields should be OR-ed
// e.g. SetOutputFields(CZR_HUMIDITY | CZR_RAWTEMP | CZR_RAWCO2);
//
// you need to set the STREAMING mode explicitly
// SetOperatingMode(CZR_STREAMING);
//
// in STREAMING mode you must parse the output of serial yourself
//
void COZIR::SetOutputFields(uint16_t fields)
{
    sprintf(buffer, "M %u", fields);
    Command(buffer);
}

// For Arduino you must read the serial yourself as
// the internal buffer of this Class cannot handle
// large output - can be > 100 bytes!!
void COZIR::GetRecentFields()
{
    Command("Q");
}

////////////////////////////////////
//
// EEPROM - USE WITH CARE
//
// SEE DATASHEET 7.2 EEPROM FOR DETAILS
//
// TODO
// - defines for addresses
// - do HILO values in one call
//
void COZIR::SetEEPROM(uint8_t address, uint8_t value)
{
    sprintf(buffer, "P %u %u", address, value);
    Command(buffer);
}

uint8_t COZIR::GetEEPROM(uint8_t address)
{
    sprintf(buffer, "p %u", address);
    return Request(buffer);
}

```

```

////////////////////////////////////
//
// COMMAND MODE
//
// read serial yourself
//
void COZIR::GetVersionSerial()
{
    Command("Y");
}

void COZIR::GetConfiguration()
{
    Command("*");
}

////////////////////////////////////
// PRIVATE

void COZIR::Command(char* s)
{
    CZR_Serial.print(s);
    CZR_Serial.print("\r\n");
}

uint16_t COZIR::Request(char* s)
{
    Command(s);
    // empty buffer
    buffer[0] = '\0';
    // read answer; there may be a 100ms delay!
    // TODO: PROPER TIMEOUT CODE.
    delay(150);
    int idx = 0;
    while(CZR_Serial.available())
    {
        buffer[idx++] = CZR_Serial.read();
    }
    buffer[idx] = '\0';
    uint16_t rv = 0;
    switch(buffer[1])
    {
        case 'T' :
            rv = atoi(&buffer[5]);
            if (buffer[4] == 1) rv += 1000;
            break;
        default :
            rv = atoi(&buffer[2]);
            break;
    }
}

```

```

    }
    return rv;
}

```

Appendix D - MRF24J40 Library.

Header File:

```

/*
 * File:    mrf24j.h
 * copyright Karl Palsson, karlp@tweak.net.au, 2011
 * modified BSD License / apache license
 */

#ifndef LIB_MRF24J_H
#define LIB_MRF24J_H

#if defined(ARDUINO) && ARDUINO >= 100 // Arduino IDE version >= 1.0
    #include "Arduino.h"
#else // older Arduino IDE versions
    #include "WProgram.h"
#endif
#include "SPI.h"

#define MRF_RXMCR 0x00
#define MRF_PANIDL 0x01
#define MRF_PANIDH 0x02
#define MRF_SADRL 0x03
#define MRF_SADRH 0x04
#define MRF_EADR0 0x05
#define MRF_EADR1 0x06
#define MRF_EADR2 0x07
#define MRF_EADR3 0x08
#define MRF_EADR4 0x09
#define MRF_EADR5 0x0A
#define MRF_EADR6 0x0B
#define MRF_EADR7 0x0C
#define MRF_RXFLUSH 0x0D
// #define MRF_Reserved 0x0E
// #define MRF_Reserved 0x0F
#define MRF_ORDER 0x10
#define MRF_TXMCR 0x11
#define MRF_ACKTMOUT 0x12
#define MRF_ESLOTG1 0x13
#define MRF_SYMTICKL 0x14
#define MRF_SYMTICKH 0x15
#define MRF_PACON0 0x16
#define MRF_PACON1 0x17
#define MRF_PACON2 0x18

```

```

//#define MRF_Reserved 0x19
#define MRF_TXBCON0 0x1A

// TXNCON: TRANSMIT NORMAL FIFO CONTROL REGISTER (ADDRESS: 0x1B)
#define MRF_TXNCON 0x1B
#define MRF_TXNTRIG 0
#define MRF_TXNSECEN 1
#define MRF_TXNACKREQ 2
#define MRF_INDIRECT 3
#define MRF_FPSTAT 4

#define MRF_TXG1CON 0x1C
#define MRF_TXG2CON 0x1D
#define MRF_ESLOTG23 0x1E
#define MRF_ESLOTG45 0x1F
#define MRF_ESLOTG67 0x20
#define MRF_TXPEND 0x21
#define MRF_WAKECON 0x22
#define MRF_FRMOFFSET 0x23
// TXSTAT: TX MAC STATUS REGISTER (ADDRESS: 0x24)
#define MRF_TXSTAT 0x24
#define TXNRETRY1 7
#define TXNRETRY0 6
#define CCAFAIL 5
#define TXG2FNT 4
#define TXG1FNT 3
#define TXG2STAT 2
#define TXG1STAT 1
#define TXNSTAT 0

#define MRF_TXBCON1 0x25
#define MRF_GATECLK 0x26
#define MRF_TXTIME 0x27
#define MRF_HSYM TMRL 0x28
#define MRF_HSYM TMRH 0x29
#define MRF_SOFTTRST 0x2A
//#define MRF_Reserved 0x2B
#define MRF_SECCON0 0x2C
#define MRF_SECCON1 0x2D
#define MRF_TXSTBL 0x2E
//#define MRF_Reserved 0x2F
#define MRF_RXSR 0x30
#define MRF_INTSTAT 0x31
#define MRF_INTCON 0x32
#define MRF_GPIO 0x33
#define MRF_TRISGPIO 0x34
#define MRF_SLPACK 0x35
#define MRF_RFCTL 0x36
#define MRF_SECCR2 0x37
#define MRF_BBREG0 0x38

```

```

#define MRF_BBREG1 0x39
#define MRF_BBREG2 0x3A
#define MRF_BBREG3 0x3B
#define MRF_BBREG4 0x3C
// #define MRF_Reserved 0x3D
#define MRF_BBREG6 0x3E
#define MRF_CCAEDTH 0x3F

#define MRF_RFCON0 0x200
#define MRF_RFCON1 0x201
#define MRF_RFCON2 0x202
#define MRF_RFCON3 0x203
#define MRF_RFCON5 0x205
#define MRF_RFCON6 0x206
#define MRF_RFCON7 0x207
#define MRF_RFCON8 0x208
#define MRF_SLPCAL0 0x209
#define MRF_SLPCAL1 0x20A
#define MRF_SLPCAL2 0x20B
#define MRF_RSSI 0x210
#define MRF_SLPCON0 0x211
#define MRF_SLPCON1 0x220
#define MRF_WAKETIMEL 0x222
#define MRF_WAKETIMEH 0x223
#define MRF_REMCNTL 0x224
#define MRF_REMCNTH 0x225
#define MRF_MAINCNT0 0x226
#define MRF_MAINCNT1 0x227
#define MRF_MAINCNT2 0x228
#define MRF_MAINCNT3 0x229
#define MRF_TESTMODE 0x22F
#define MRF ASSOEADR1 0x231
#define MRF ASSOEADR2 0x232
#define MRF ASSOEADR3 0x233
#define MRF ASSOEADR4 0x234
#define MRF ASSOEADR5 0x235
#define MRF ASSOEADR6 0x236
#define MRF ASSOEADR7 0x237
#define MRF ASSOSADR0 0x238
#define MRF ASSOSADR1 0x239
#define MRF_UPNONCE0 0x240
#define MRF_UPNONCE1 0x241
#define MRF_UPNONCE2 0x242
#define MRF_UPNONCE3 0x243
#define MRF_UPNONCE4 0x244
#define MRF_UPNONCE5 0x245
#define MRF_UPNONCE6 0x246
#define MRF_UPNONCE7 0x247
#define MRF_UPNONCE8 0x248
#define MRF_UPNONCE9 0x249

```

```

#define MRF_UPNONCE10 0x24A
#define MRF_UPNONCE11 0x24B
#define MRF_UPNONCE12 0x24C

#define MRF_I_RXIF 0b00001000
#define MRF_I_TXNIF 0b00000001

typedef struct _rx_info_t{
    uint8_t frame_length;
    uint8_t rx_data[116]; //max data length = (127 aMaxPHYPacketSize - 2 Frame
control - 1 sequence number - 2 panid - 2 shortAddr Destination - 2 shortAddr
Source - 2 FCS)
    uint8_t lqi;
    uint8_t rssi;
} rx_info_t;

/**
 * Based on the TXSTAT register, but "better"
 */
typedef struct _tx_info_t{
    uint8_t tx_ok:1;
    uint8_t retries:2;
    uint8_t channel_busy:1;
} tx_info_t;

class Mrf24j
{
public:
    Mrf24j(int pin_reset, int pin_chip_select, int pin_interrupt, int
pin_scklock);
    void reset(void);
    void init(void);

    byte read_short(byte address);
    byte read_long(word address);

    void write_short(byte address, byte data);
    void write_long(word address, byte data);

    word get_pan(void);
    void set_pan(word panid);

    void address16_write(word address16);
    word address16_read(void);

    void set_interrupts(void);

    void set_promiscuous(boolean enabled);

    /**

```



```

    * Set the channel, using 802.15.4 channel numbers (11..26)
    */
    void set_channel(byte channel);

    void rx_enable(void);
    void rx_disable(void);

    /** If you want to throw away rx data */
    void rx_flush(void);

    rx_info_t * get_rxinfo(void);

    tx_info_t * get_txinfo(void);

    uint8_t * get_rxbuf(void);

    int rx_datalength(void);

    void set_ignoreBytes(int ib);

    /**
     * Set bufPHY flag to buffer all bytes in PHY Payload, or not
     */
    void set_bufferPHY(boolean bp);

    boolean get_bufferPHY(void);

    /**
     * Set PA/LNA external control
     */
    void set_palna(boolean enabled);

    void send16(word dest16, char * data);

    void interrupt_handler(void);

    void check_flags(void (*rx_handler)(void), void (*tx_handler)(void));

private:
    int _pin_reset;
    int _pin_cs;
    int _pin_int;
    int _pin_sck;
    static SPISettings spiSettings;
};

#endif /* LIB_MRF24J_H */

```

C++ File:

```
/**
 * mrf24j.cpp, Karl Palsson, 2011, karlp@tweak.net.au
 * modified bsd license / apache license
 */

#include "mrf24j.h"

// aMaxPHYPacketSize = 127, from the 802.15.4-2006 standard.
static uint8_t rx_buf[127];

// essential for obtaining the data frame only
// bytes_MHR = 2 Frame control + 1 sequence number + 2 panid + 2 shortAddr
// Destination + 2 shortAddr Source
static int bytes_MHR = 9;
static int bytes_FCS = 2; // FCS length = 2
static int bytes_nodata = bytes_MHR + bytes_FCS; // no_data bytes in PHY
// payload, header length + FCS

static int ignoreBytes = 0; // bytes to ignore, some modules behaviour.

static boolean bufPHY = false; // flag to buffer all bytes in PHY Payload, or
// not

volatile uint8_t flag_got_rx;
volatile uint8_t flag_got_tx;

static rx_info_t rx_info;
static tx_info_t tx_info;

/**
 * Constructor MRF24J Object.
 * @param pin_reset, @param pin_chip_select, @param pin_interrupt
 */
SPISettings Mrf24j::spiSettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);

Mrf24j::Mrf24j(int pin_reset, int pin_chip_select, int pin_interrupt, int
pin_sck) {
    _pin_reset = pin_reset;
    _pin_cs = pin_chip_select;
    _pin_int = pin_interrupt;
    _pin_sck = pin_sck;

    pinMode(_pin_reset, OUTPUT);
    pinMode(_pin_cs, OUTPUT);
    pinMode(_pin_int, INPUT);
    pinMode(_pin_sck, OUTPUT);

    //SPI.setBitOrder(MSBFIRST) ;
}
```

```

    //SPI.setDataMode(SPI_MODE0);

    SPI.begin();
}

void Mrf24j::reset(void) {
    digitalWrite(_pin_reset, LOW);
    delay(10); // just my gut
    digitalWrite(_pin_reset, HIGH);
    delay(20); // from manual
}

byte Mrf24j::read_short(byte address) {
    SPI.beginTransaction(spiSettings); //updated
    digitalWrite(_pin_cs, LOW);
    // 0 top for short addressing, 0 bottom for read
    SPI.transfer(address<<1 & 0b01111110);
    byte ret = SPI.transfer(0x00);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction(); //updated
    return ret;
}

byte Mrf24j::read_long(word address) {
    SPI.beginTransaction(spiSettings); //updated
    digitalWrite(_pin_cs, LOW);
    byte ahigh = address >> 3;
    byte alow = address << 5;
    SPI.transfer(0x80 | ahigh); // high bit for long
    SPI.transfer(alow);
    byte ret = SPI.transfer(0);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction(); //updated
    return ret;
}

void Mrf24j::write_short(byte address, byte data) {
    SPI.beginTransaction(spiSettings); //updated
    digitalWrite(_pin_cs, LOW);
    // 0 for top short address, 1 bottom for write
    SPI.transfer((address<<1 & 0b01111110) | 0x01);
    SPI.transfer(data);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction(); //updated
}

void Mrf24j::write_long(word address, byte data) {
    SPI.beginTransaction(spiSettings); //updated
    digitalWrite(_pin_cs, LOW);

```

```

    byte ahigh = address >> 3;
    byte alow = address << 5;
    SPI.transfer(0x80 | ahigh); // high bit for long
    SPI.transfer(alow | 0x10); // last bit for write
    SPI.transfer(data);
    digitalWrite(_pin_cs, HIGH);
    SPI.endTransaction(); //updated
}

word Mrf24j::get_pan(void) {
    byte panh = read_short(MRF_PANIDH);
    return panh << 8 | read_short(MRF_PANIDL);
}

void Mrf24j::set_pan(word panid) {
    write_short(MRF_PANIDH, panid >> 8);
    write_short(MRF_PANIDL, panid & 0xff);
}

void Mrf24j::address16_write(word address16) {
    write_short(MRF_SADRH, address16 >> 8);
    write_short(MRF_SADRL, address16 & 0xff);
}

word Mrf24j::address16_read(void) {
    byte a16h = read_short(MRF_SADRH);
    return a16h << 8 | read_short(MRF_SADRL);
}

/**
 * Simple send 16, with acks, not much of anything.. assumes src16 and local pan
 * only.
 * @param data
 */
void Mrf24j::send16(word dest16, char * data) {
    byte len = strlen(data); // get the length of the char* array
    int i = 0;
    write_long(i++, bytes_MHR); // header length
    // +ignoreBytes is because some module seems to ignore 2 bytes after the
    header?!.
    // default: ignoreBytes = 0;
    write_long(i++, bytes_MHR+ignoreBytes+len);

    // 0 | pan compression | ack | no security | no data pending | data frame[3
    bits]
    write_long(i++, 0b01100001); // first byte of Frame Control
    // 16 bit source, 802.15.4 (2003), 16 bit dest,
    write_long(i++, 0b10001000); // second byte of frame control
    write_long(i++, 1); // sequence number 1

```

```

word panid = get_pan();

write_long(i++, panid & 0xff); // dest panid
write_long(i++, panid >> 8);
write_long(i++, dest16 & 0xff); // dest16 low
write_long(i++, dest16 >> 8); // dest16 high

word src16 = address16_read();
write_long(i++, src16 & 0xff); // src16 low
write_long(i++, src16 >> 8); // src16 high

// All testing seems to indicate that the next two bytes are ignored.
//2 bytes on FCS appended by TXMAC
i+=ignoreBytes;
for (int q = 0; q < len; q++) {
    write_long(i++, data[q]);
}
// ack on, and go!
write_short(MRF_TXNCON, (1<<MRF_TXNACKREQ | 1<<MRF_TXNTRIG));
}

void Mrf24j::set_interrupts(void) {
    // interrupts for rx and tx normal complete
    write_short(MRF_INTCON, 0b11110110); //0b11110110-previous config
}

/** use the 802.15.4 channel numbers..
 */
void Mrf24j::set_channel(byte channel) {
    write_long(MRF_RFCON0, (((channel - 11) << 4) | 0x03));
}

void Mrf24j::init(void) {

    /**/ Uncomment to set Reset at start of module
    write_short(MRF_SOFT_RST, 0x7); // from manual
    while (read_short(MRF_SOFT_RST) & 0x7 != 0) {
        ; // wait for soft reset to finish
    }*/

    write_short(MRF_PACON2, 0x98); // - Initialize FIFOEN = 1 and TXONTS = 0x6.
    write_short(MRF_TXSTBL, 0x95); // - Initialize RFSTBL = 0x9.

    write_long(MRF_RFCON0, 0x03); // - Initialize RFOP = 0x03.
    write_long(MRF_RFCON1, 0x01); // - Initialize VCOOPT = 0x02.
    write_long(MRF_RFCON2, 0x80); // - Enable PLL (PLLEN = 1).
    write_long(MRF_RFCON6, 0x90); // - Initialize TXFIL = 1 and 20MRECVR = 1.
    write_long(MRF_RFCON7, 0x80); // - Initialize SLPCLKSEL = 0x2 (100 kHz
Internal oscillator).
    write_long(MRF_RFCON8, 0x10); // - Initialize RFVCO = 1.

```

```

    write_long(MRF_SLPCON1, 0x21); // - Initialize CLKOUTEN = 1 and SLPCLKDIV =
0x01.

    // Configuration for nonbeacon-enabled devices (see Section 3.8
"Beacon-Enabled and Nonbeacon-Enabled Networks"):
    write_short(MRF_BBREG2, 0x80); // Set CCA mode to ED
    write_short(MRF_CCAEDTH, 0x60); // - Set CCA ED threshold.
    write_short(MRF_BBREG6, 0x40); // - Set appended RSSI value to RXFIFO.
    set_interrupts();
    set_channel(11);
    // max power is by default.. just leave it...
    // Set transmitter power - See "REGISTER 2-62: RF CONTROL 3 REGISTER
(ADDRESS: 0x203)".
    write_short(MRF_RFCTL, 0x04); // - Reset RF state machine.
    write_short(MRF_RFCTL, 0x00); // part 2
    delay(1); // delay at least 192usec
}

/**
 * Call this from within an interrupt handler connected to the MRFs output
 * interrupt pin. It handles reading in any data from the module, and letting
it
 * continue working.
 * Only the most recent data is ever kept.
 */
void Mrf24j::interrupt_handler(void) {
    uint8_t last_interrupt = read_short(MRF_INTSTAT);
    if (last_interrupt & MRF_I_RXIF) {
        flag_got_rx++;
        // read out the packet data...
        noInterrupts();
        rx_disable();
        // read start of rxfifo for, has 2 bytes more added by FCS. frame_length
= m + n + 2
        uint8_t frame_length = read_long(0x300);

        // buffer all bytes in PHY Payload
        if(bufPHY){
            int rb_ptr = 0;
            for (int i = 0; i < frame_length; i++) { // from 0x301 to (0x301 +
frame_length -1)
                rx_buf[rb_ptr++] = read_long(0x301 + i);
            }
        }

        // buffer data bytes
        int rd_ptr = 0;
        // from (0x301 + bytes_MHR) to (0x301 + frame_length - bytes_nodata - 1)
        for (int i = 0; i < rx_datalength(); i++) {
            rx_info.rx_data[rd_ptr++] = read_long(0x301 + bytes_MHR + i);

```

```

    }

    rx_info.frame_length = frame_length;
    // same as datasheet 0x301 + (m + n + 2) <-- frame_length
    rx_info.lqi = read_long(0x301 + frame_length);
    // same as datasheet 0x301 + (m + n + 3) <-- frame_length + 1
    rx_info.rssi = read_long(0x301 + frame_length + 1);

    rx_enable();
    interrupts();
}
if (last_interrupt & MRF_I_TXNIF) {
    flag_got_tx++;
    uint8_t tmp = read_short(MRF_TXSTAT);
    // 1 means it failed, we want 1 to mean it worked.
    tx_info.tx_ok = !(tmp & ~(1 << TXNSTAT));
    tx_info.retries = tmp >> 6;
    tx_info.channel_busy = (tmp & (1 << CCAFAIL));
}
}

/**
 * Call this function periodically, it will invoke your nominated handlers
 */
void Mrf24j::check_flags(void (*rx_handler)(void), void (*tx_handler)(void)){
    if (flag_got_rx) {
        flag_got_rx = 0;
        rx_handler();
    }
    if (flag_got_tx) {
        flag_got_tx = 0;
        tx_handler();
    }
}

/**
 * Set RX mode to promiscuous, or normal
 */
void Mrf24j::set_promiscuous(boolean enabled) {
    if (enabled) {
        write_short(MRF_RXMCR, 0x01);
    } else {
        write_short(MRF_RXMCR, 0x00);
    }
}

rx_info_t * Mrf24j::get_rxinfo(void) {
    return &rx_info;
}

```

```

tx_info_t * Mrf24j::get_txinfo(void) {
    return &tx_info;
}

uint8_t * Mrf24j::get_rxbuf(void) {
    return rx_buf;
}

int Mrf24j::rx_datalength(void) {
    return rx_info.frame_length - bytes_nodata;
}

void Mrf24j::set_ignoreBytes(int ib) {
    // some modules behaviour
    ignoreBytes = ib;
}

/**
 * Set bufPHY flag to buffer all bytes in PHY Payload, or not
 */
void Mrf24j::set_bufferPHY(boolean bp) {
    bufPHY = bp;
}

boolean Mrf24j::get_bufferPHY(void) {
    return bufPHY;
}

/**
 * Set PA/LNA external control
 */
void Mrf24j::set_palna(boolean enabled) {
    if (enabled) {
        write_long(MRF_TESTMODE, 0x07); // Enable PA/LNA on MRF24J40MB module.
    } else {
        write_long(MRF_TESTMODE, 0x00); // Disable PA/LNA on MRF24J40MB module.
    }
}

void Mrf24j::rx_flush(void) {
    write_short(MRF_RXFLUSH, 0x01);
}

void Mrf24j::rx_disable(void) {
    write_short(MRF_BBREG1, 0x04); // RXDECINV - disable receiver
}

void Mrf24j::rx_enable(void) {
    write_short(MRF_BBREG1, 0x00); // RXDECINV - enable receiver
}

```