Bore Pump Control

Architecture Notebook

**Change log**

This table lists the changes since LCOM.

|  |  |  |
| --- | --- | --- |
| Date | Author | Notes |
| 2020-03-30 | Andrew | Initial commit. |
| 2020-04-22 | David | Noted class A device constraints and added controller class diagram. |
| 2020-05-04 | David | Added details of device message formats. Updated class diagram. |
| 2020-05-05 | David | Added subsystem section to describe the dashboard engine, rules engine, TTN, etc. |
| 2020-05-07 | David | Updated the system diagram to show TTN and TB as separate processes. |
| 2020-05-10 | David | Updated mechanisms section to take into account what we've learned about TTN/LoRaWAN. |
| 2020-05-22 | David | Updated the message flow in the DFD. |
| 2020-06-03 | David | Updated diagrams, status msg fields, added section on firmware logic. Made heading styles consistent. Removed "proposed" from the filename. |
| 2020-06-15 | David | Added title page and change log. |
| 2020-08-07 | David | Added timeout details to pump control message and mention bore low level signal in sections about starting/stopping the pump. |
| 2020-09-14 | David | Updated the no-ack messaging info and made some text more generic around alarm signals. |
| 2020-10-18 | Andrew | Updated Use Case Diagram to better reflect implemented system. |

# Purpose

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation.

# Architectural goals and philosophy

The system is designed around providing a simple interface for allowing the manual and automatic remote control of a bore pump on a farm. The sponsor already has in place a large amount of infrastructure which we will use to implement this new system.

Given the remote-control nature of the project there are several parts to the physical architecture that will need to communicate with one another. Communication needs to be long range, and available in places where traditional infrastructure such as mobile signal or WiFi are not available. The sponsor has already put in place a LoRaWAN Network to handle this communication.

The control of the bore pump needs to be handled by low-powered device that can be easily shielded from the environment where the bore pump operates. The Sponsor has elected to use an Adafruit Feather board, as it can connect to their existing LoRaWAN network, and is in place for several their other projects.

The system needs to interface with the existing farm control network; Thingsboard. Thingsboard is already in place where the sponsor uses it to track a number of other sensors around the farm. This will allow the operator to access many of the farm’s remote-controlled functions from one application. Thingsboard has rules and scripting engines that will allow us to use it to handle the automation logic, the logging of events and build a front-end for the user to interact with.

All interactions between the pump and the dashboard will be logged. This will allow users to get a visual representation of pump usage.

The system may become part of a larger system that takes readings from other sensors to determine when the bore pump should be used. While this is outside of the scope of this project, it is worth keeping in mind to ensure that code is clear and well commented for future users to maintain.

## Assumptions and dependencies

* ThingsBoard: This is the existing infrastructure used by the sponsor. This project must interface with existing sensors connected to Thingsboard and centralize control and view of logs through it.
* LoRaWAN: This is the communication method supplied by the sponsor to allow communication between the controller and Thingsboard. This introduces a number of constraints (covered later in this doc) that must be adhered to.
* AdaFruit FeatherBoard: This is the hardware that controls the bore pump. The sponsor has chosen the Feather form factor as their standard for embedded projects. This project must control the pump using software run on this board.
* C++: The Feather is programmed with C++.
* JavaScript: Thingsboard and Things Network are programmed with JavaScript.

## Architecturally significant requirements

* The system must communicate over the LoRaWAN network and within the limits of that network.
* Logging, pump control and pump automation must be built in Thingsboard.
* Pump must be controlled by Adafruit Featherboard.

## Decisions, constraints, and justifications

* LoRaWAN Constraint: Pump to Thingsboard messages must be small and infrequent. The fair use policy of The Things Network allows each node an average of 3 minutes of airtime per day for uplink messages (node to gateway) and 10 downlink messages per day (gateway to node).
* LoRaWAN Constraint: The pump controller will run as a class A end node meaning it can only receive messages directly after sending a status report. This introduces a delay between the user issuing a command, the command message being sent, and the result of the command being known.
* Pump and Thingsboard must be able to communicate without message acknowledgement. This is to keep within the above restrictions.
* The pump has auto cutoff mechanisms for low bore, and high tank level. These will remain in place as a backup to the automation being introduced by this system.
* Pump status cannot be tracked in real time. The pump is subject to other sensors and manual intervention which may mean the status indicated by the dashboard is incorrect until the new status comes from the pump.

# Architectural Mechanisms

## Architectural Mechanism 1

**No ACK Messaging:** Uplink telemetry messages will be sent without expecting an acknowledgement from the Things Network.

Downlink command messages do have the confirmed flag set so Things Network will keep sending them after each uplink until the pump controller acknowledges receipt.

This was done because during development it was difficult to get a downlink command message received on the first attempt, except on site at the Orange Agricultural Institute.

## Architectural Mechanism 2

**Replace Messages:** If the rules engine decides a command must be sent to the pump controller before the previously queued command has been sent, the new command will *replace* the previously queued command rather than being added to the queue and both messages sent. This provides two benefits:

1. It simplifies the business logic that must be encoded within the rules engine because the server side does not need to remember if it has already queued a pump command before sending another of the same type.
2. It reduces the number of downlink messages. For example, if the rules engine determined the pump should be switched on, and the user manually chooses to switch the pump off after that, only the pump off command will be sent to the pump controller.

The pump controller will replace uplink messages if a message was scheduled to be sent, and another message is scheduled to be sent before the transmission window opens. In this case only the second message will be sent.

## Architectural Mechanism 3

**Timed Events:** Due to the nature of LoRaWAN the server-side application cannot request status updates or have real-time knowledge of the state of the pump controller. Having the pump controller send status updates at regular intervals allows the server-side application to keep relatively up-to date with the pump controller status..

## Architectural Mechanism 4

**Thingsboard Logic Engine:** This is the mechanism by which automation and logic rules will be applied to the data collected in the system.

## Architectural Mechanism 5

**Thingsboard Dashboard Engine:** This is the mechanism by which we will implement a ui for the user to interact with, control the pump and view logs/data.

## Architectural Mechanism 6

**Bit Flags:** Status messages from the LoRaWAN need to be kept as small as possible. To achieve this, messages will be sent as a series of bit (on/off) flags. This keeps messages short and allows us to offload work of interpreting to the server which has more power.

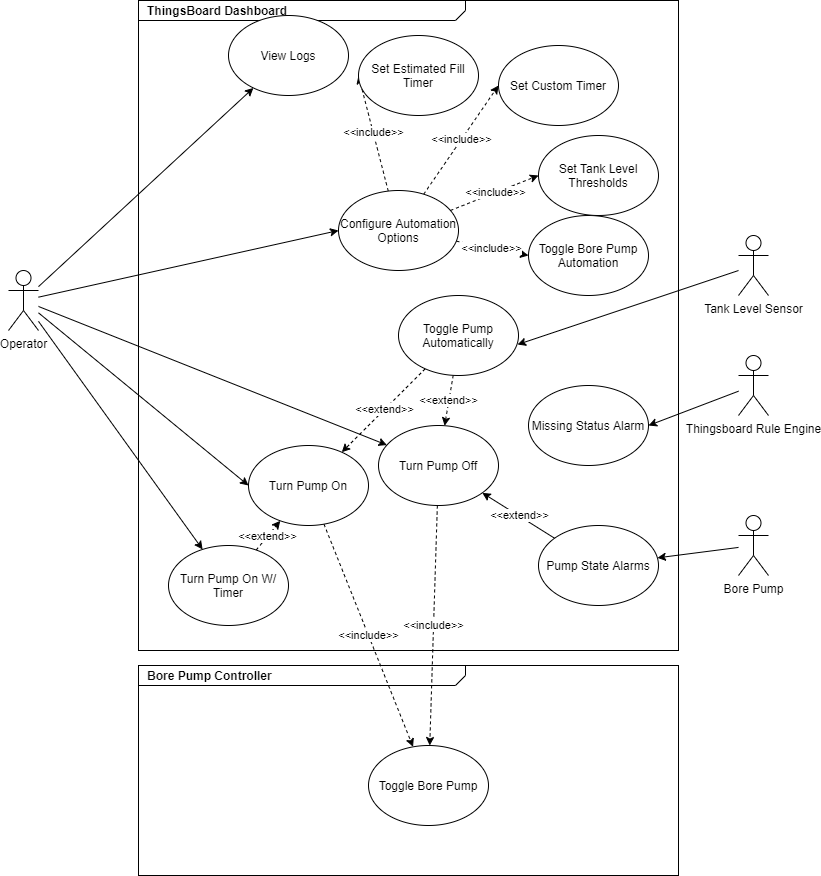
## Architectural Mechanism 7

**Message on state change:** The pump controller will attempt to send a status message as soon as its state changes such as the pump being turned on or off, or one of the input pins changing state. This is to minimise the time the dashboard is out of sync with the real state of the controller. For example, if the user switches the pump on manually, that command will not be sent to the pump controller until after the pump controller uplinks a status message. In this case the dashboard will think the pump is not running because the status message was sent before the pump controller received the command. So the pump controller will attempt to send another status message after switching the pump on so the dashboard is informed of the new state as soon as possible.

# Architectural views

## Use Cases

See use case Doc for Detailed Descriptions.



## Logical View

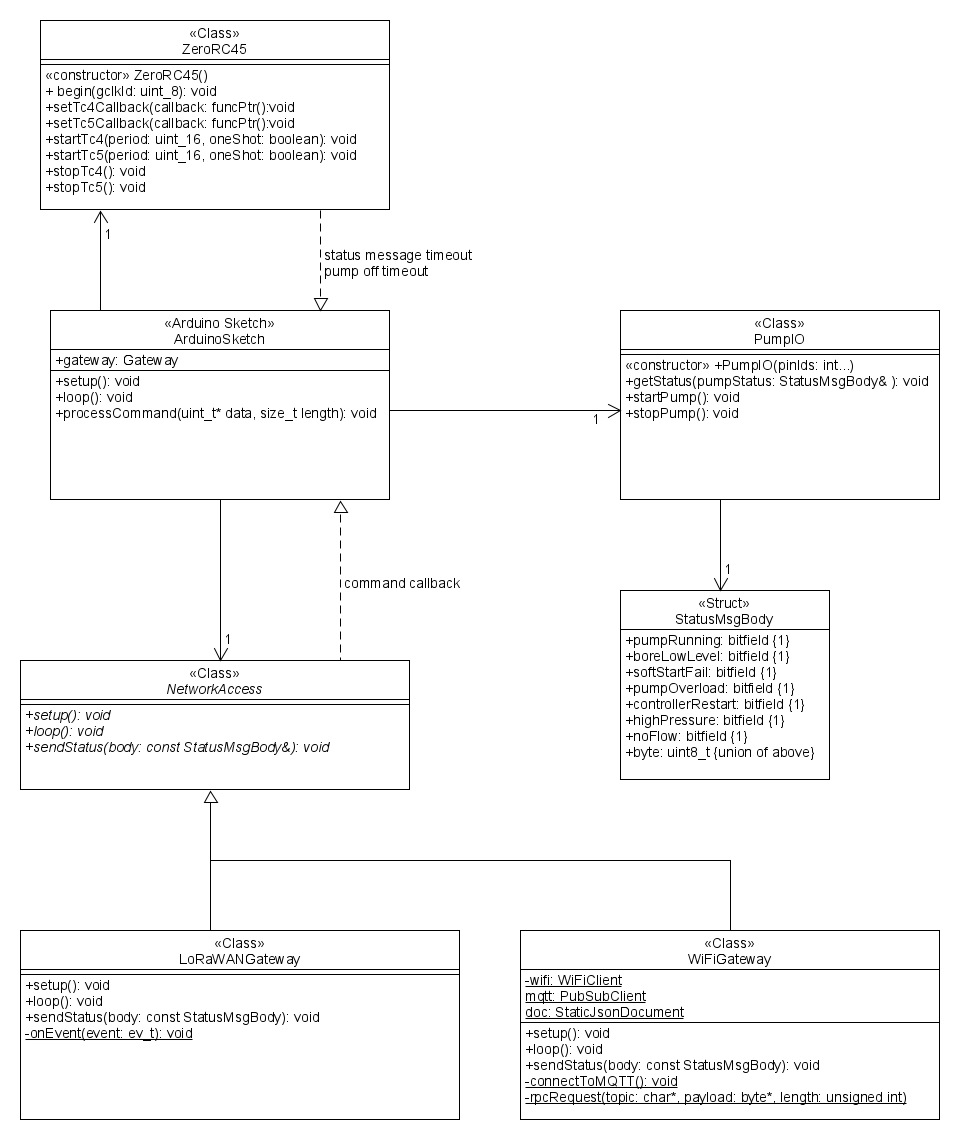
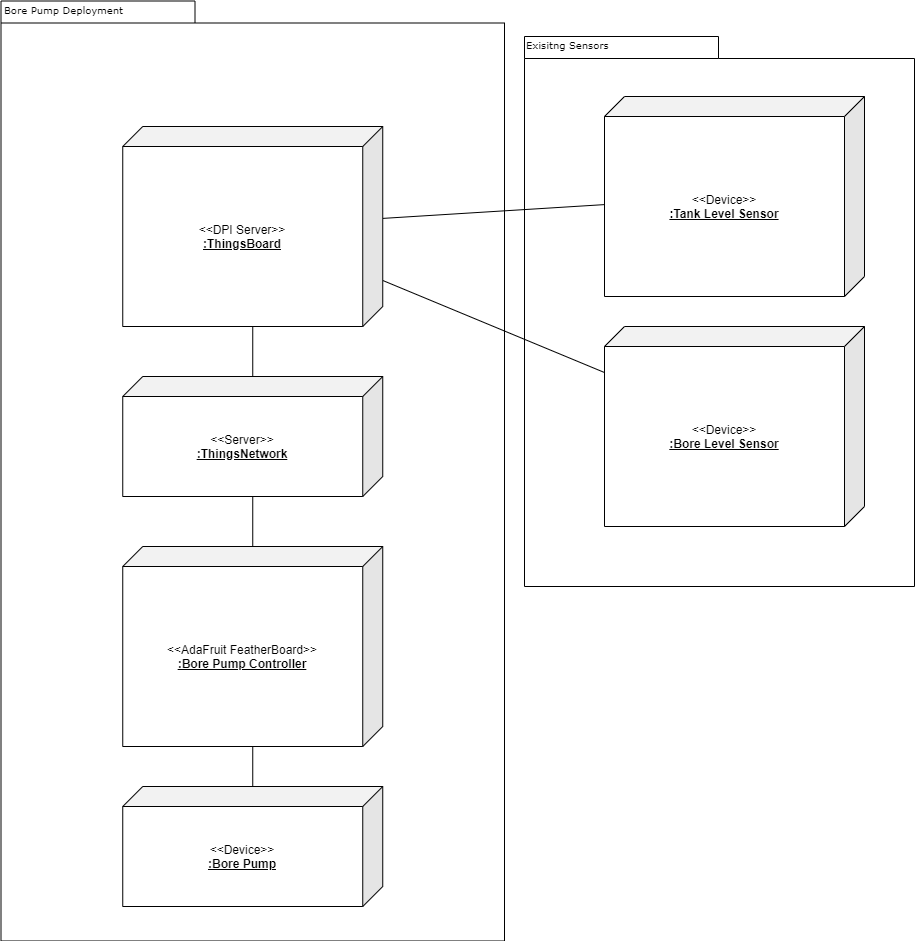
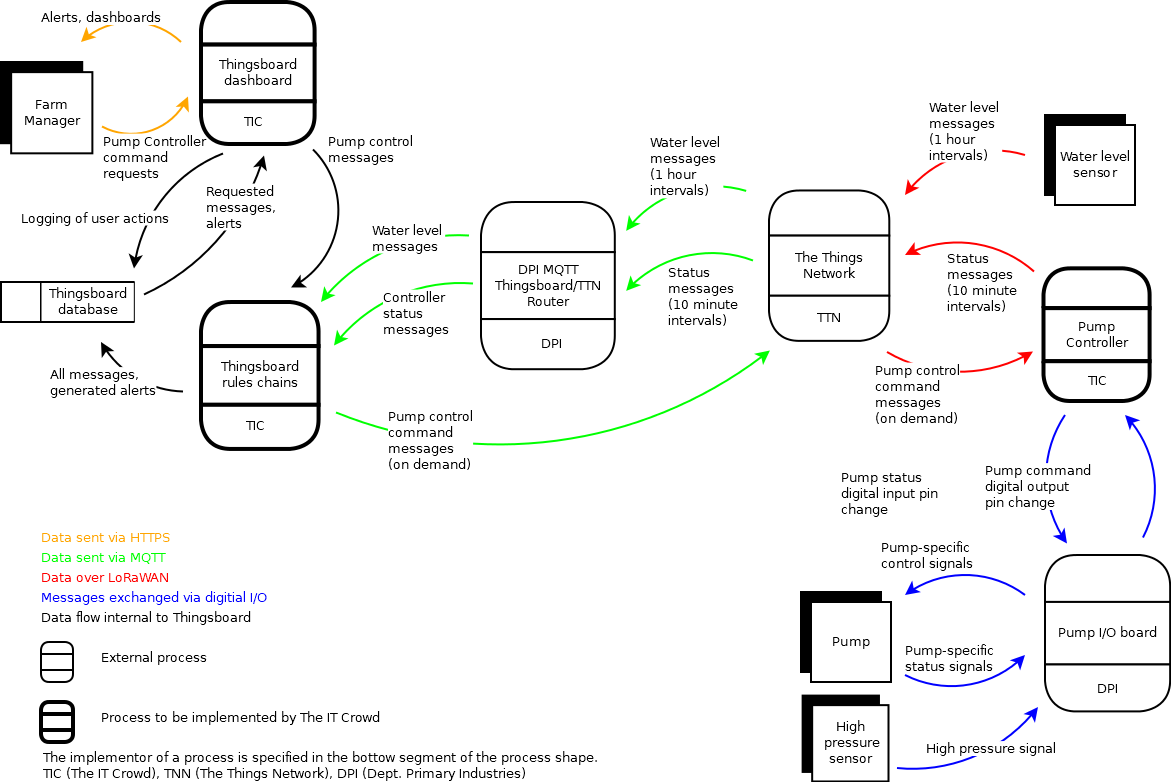


Figure 2 Pump Controller Class Diagram

## Physical View



## Process View



## Data Model

This section details the message formats from/to the devices. The messages are transformed multiple times as they travel between the devices, the LoRaWAN network, The Things Network, and Thingsboard. This section will only detail the messages as the devices see them.

### Water level message

The water level sensor sends an 8-byte message. DPI have supplied us with the algorithm required to decode the message.

|  |  |
| --- | --- |
| Bytes | Content |
| 0,1 | Unknown. |
| 2 | 0 = telemetry message, messages with other values will be ignored. |
| 3,4 | Depth in metres. Interpret the bytes as a 16-bit big-endian integer transformed as follows. Round the final result to 2 decimal places.  depth = (msgByte[3] << 8) + msgByte[4]; // Make the 16-bit BE integer. depth = (depth - 1638.3) \* (5/13106.4); // Apply DPI formula. |
| 5,6 | Water temperature in deg C. Bytes give a 16-bit big-endian integer that is transformed as follows. Round the final result to 2 decimal places.  tempC = (msgByte[5] << 8) + msgByte[6]; // Make the 16-bit BE integer. tempC = tempC \* 0.001; // Correction factor provided by DPI. |
| 7 | Battery voltage. A single-byte integer, transformed as follows. Round the final result to 2 decimal places.  volts = msgByte[7] \* 0.1; // Scale value supplied by DPI. |

### Pump controller status message

The pump controller sends a status message every *n* minutes. It is a single byte with the following bitflags, ordered starting with the least-significant byte.

|  |  |  |
| --- | --- | --- |
| Bit | Flag | Description |
| 0 | pumpRunning | 0 = pump not running, 1 = pump running. |
| 1 | boreLowLevel | 1 = low input pressure, meaning the bore water level is low. |
| 2 | softStartFail | The pump is connected to a motor controller and this is probably a signal that the motor soft start feature is not working. |
| 3 | pumpOverload | This is probably another signal from the motor controller rather than the pump itself. |
| 4 | controllerRestart | This flag is set in the first status message sent by the pump controller. It allows the server-side app(s) to track restarts. |
| 5 | highPressure | 1 = The high pressure sensor in the tank is showing the water level is high. |
| 6 | noFlow | Don’t know. |
| 7 | Reserved | Not used yet. |

### Pump Controller control message

The pump controller control message is sent from Thingsboard to the pump controller to switch the pump on or off. It is one or three bytes. The first byte has the following bitflags, ordered starting with the least-significant byte.

There will probably be another bit flag to signal a timeout value follows the message byte. We have not implemented that yet.

|  |  |  |
| --- | --- | --- |
| Bit | Flag | Description |
| 0 | pumpState | 0 = stop pump, 1 = start pump. |
| 1 | timeout | This flag is only valid when pumpState = 1.  0 = no timeout, 1 = timeout follows in next two bytes.  The timeout is a 16-bit unsigned integer with the most significant byte first. It is interpreted as minutes. |
| 2-7 | Reserved | Not used yet. |

Example messages:

0x00 Switch pump off.  
0x01 Switch pump on, no timeout.  
0x03 0x00 0x78 Switch pump on, switch it off in 120 minutes if it is still running.  
0x03 0x01 0x0E Switch pump on, switch it off in 270 minutes if it is still running.

# Subsystem Descriptions

This section describes each of the subsystems involved in the Bore Pump Control project.

## The Things Network

The Things Network is a cloud-based gateway that allows messages to be routed between LoRaWAN gateways and traditional server-side applications. It is essentially a message conversion and routing engine. It allows the user to define message conversion and validation functions so message payloads can be converted between json and binary formats, or it can just accept/transmit the binary payload, wrapped in a json object with other metadata such as device ids, timestamps, etc.

The Things Network does not play any application level role in the Bore Pump Control project – it is only used to get messages from and to the edge devices – the tank level sensor and the bore pump controller.

## DPI MQTT Router

Both The Things Network and Thingsboard use MQTT for sending and receiving messages. For telemetry messages – ie uplink messages from nodes – The Things Network publishes them to an MQTT queue in its internal broker from which they can be read. Thingsboard requires incoming telemetry messages to be published to an MQTT queue in its internal broker. The DPI MQTT Router is a “middleman” written by DPI that listens for messages published to The Things Network broker and publishes them to the Thingsboard broker.

## Thingsboard Rules Engine

The Thingsboard rules engine runs messages through a series of rules chains to record, transform, or otherwise act on those messages. Examples of actions are “record telemetry to database”, “send a message to device”, “raise an alarm”, “get device attribute values”.

One rule chain can call another, forwarding the current message into the new rule chain. This allows business logic to be separated into discrete rules chains, analogous to functions in a normal programming environment. This also allows the ‘root rule chain’ that is invoked for every message to be kept relatively simple by moving complicated chains of logic into separate rule chains.

## Thingsboard Dashboard Engine

The Thingsboard dashboard engine allows developers to create dashboards using various ‘widgets’ that can present information received from devices and can be used to initiate the transmission of a message to a device.

## The Pump Controller

The Pump Controller is an embedded controller that reads the state of the bore pump via the Pump I/O Board and reports this status to the server-side application for analysis and presentation to the user. It also listens for requests from the server-side application to switch the pump on or off based upon business rules evaluated in the Thingsboard rules engine.

## The Pump I/O Board

The pump I/O board is an interface between the pump controller and the actual bore pump and its related electronics. It translates the various status signals to a digital on/off signal suitable for the bore pump controller to read. It also translates the output signal from the bore pump controller to switch the pump on or off to the required actions to make this happen.

We have very little information on this subsystem or how it will work so the above is our working assumptions at present. It is abstracted behind the PumpIO class in the pump controller firmware.

## High Pressure Sensor

There is a high pressure sensor installed in the water tank which is independent from the bore pump. It activates when the water level in the tank is high enough that the pump should not run. This sensor will be connected to the I/O board and this supplies the high pressure signal.

# Firmware Logic

This section describes the logic coded into the firmware for responding to downlink commands and signals from the I/O board.

## Downlink commands

Downlink commands are processed synchronously with the normal thread of control in the loop function and are acted upon as soon as they are received. Due to the nature of the LoRaWAN stack, even though the commands look like they arrive in an asynchronous callback function they are actually processed in the same thread as the main loop function. This is because the LoRaWAN stack polls the RF module during the os\_runloop\_once function (called by the main loop function) and makes the callback during the poll when all the downlink data have arrived.

The WiFi version of the firmware also has synchronous callbacks, but via a different path. To emulate the “downlink only after an uplink” behaviour of LoRaWAN class A devices, the WiFi stack holds any downlink command it receives via MQTT until the main loop function calls the sendStatus method, at which point it calls the sketch processCommand callback function. The WiFi stack also emulates the LoRaWAN replacement of downlink messages by only giving the most recently received downlink message to the callback function.

In either case the callback happens synchronously with the main loop function so there should not be any race conditions with the status flags when processing a downlink command. The actual input lines may change state but the firmware won’t see that until after the command has been processed.

### Switching the pump on

The pump control line will be brought high as soon as a downlink command requesting the pump be switched on is received, unless any of the input signal lines are low. In that case the pump control line is left low.

### Switching the pump off

The pump control line will always be brought low as soon as a downlink command requesting the pump be switched off is received.

The pump will also be switched off at the end of the period defined by a timeout interval sent with a pump on control message.

## Uplink status messages

### Message Transmission

#### LoRaWAN uplink messages

Uplink messages can only be sent if the pump controller has joined the LoRaWAN network.

An uplink message will be discarded if the LoRaWAN stack is already sending a message and is in the 2-second tx/rx window. The worst-case scenario from this is the next status message is 10 minutes away. That was the original design of the system so nothing has been lost.

If a second uplink message is sent before the previous message has started transmitting, the second message will replace the previous message.

#### WiFi uplink messages

Uplink messages can only be sent if the pump controller has joined a WiFi network and can connect to the Thingsboard MQTT broker.

The WiFi stack will send all messages given to it without delay.

### Scheduled uplink status messages

Status commands are sent every 10 minutes. The firmware will always attempt to send this message regardless of how recently another status message was sent due to state changes to the I/O lines.

### Pump start/stop status messages

A status message will be sent as soon as the pump controller changes the state of the pump control line.

### Input line state changes

A status message will be sent as soon as any input line changes state. This might have to be revisited depending on how often the input lines change state in practice.

### Other effects of input line changes

If any of the input signal lines go low while the pump control line is high, the pump control line will be brought low to stop the pump. A status message will be sent to reflect the change of state.

# The Things Network LoRaWAN fair use policy

At the time of writing The Things Network has a fair use policy that gives each device 3 minutes air time “on average” for uplink messages per day and allows 10 downlink messages per day to the device. The policy also states “A good *goal* is to keep the application payload under 12 bytes, and the interval between messages at least several minutes.” Emphasis is ours.

The Things Network console app provides an estimate of the air time required for each message it receives. The single byte uplink message from the pump controller is estimated to take between 25 and 52 milliseconds of airtime.

*We are assuming this estimate takes variables such as the frequency band and spreading factor into account, and this is what is causing the 100% difference between lower and upper estimates.*

This gives at worst 19 messages per second of air time thus allowing for 3,420 messages in the 3 minutes of air time per day.

We are sending 6 messages per hour, so 144 messages per day.

We are well within The Things Network fair use policy, both in terms of air time and the goal of several minutes between messages. There is a lot of headroom for adding new features.

The downlink message count depends on the stability of tank level readings and the speed of the pump, as well as the operator’s use of the manual control. The fact that tank level readings are hourly, and that scheduled but unsent downlink messages are *replaced* by new messages rather than having both sent should mitigate the risk of sending too many downlink messages per day.

Additionally, the pump is already working fine without this system in place so losing a downlink message should not be a disaster.