Stove Use Monitor Arduino sprint

**SUM requirements:**

The following requirements were set

1. Battery operated monitoring
2. Portable (System fitted to a cookstove and movable)
3. Measure temperature of cookstove, to measure when in use, fuelling patterns, temperature/cooking profiles.
4. Data recording: Either wireless or through sd card; ideally both
5. Maintenance: Battery change every 2 weeks

**Feature List for sprint:**

1. Battery operated monitoring (Done)
2. Sense temperature from a thermocouple (Done)
3. Transfer data over GSM to a remote server (Done)
4. Power measurements (2 weeks operation) (Done)

**Features outside of scope for sprint:**

1. Portability/packaging
2. Writing/buffering to SD card
3. Web interface/Data representation/displaying
4. Deployment on cookstove

**Overview**:

* A battery powered Arduino MKR1400 takes a temperature reading from a K-type thermocouple (via a MAX31850 breakout board). See figure 1.
* If the temperature is different from a previous temperature reading by more than 0.5C, the temperature is transmitted using the onboard GSM chip to a remote MQTT broker.
* A remote server subscribes to the MQTT broker and saves the data sent by the MKR1400 to a CSV file (node-red is used for the server-side application).

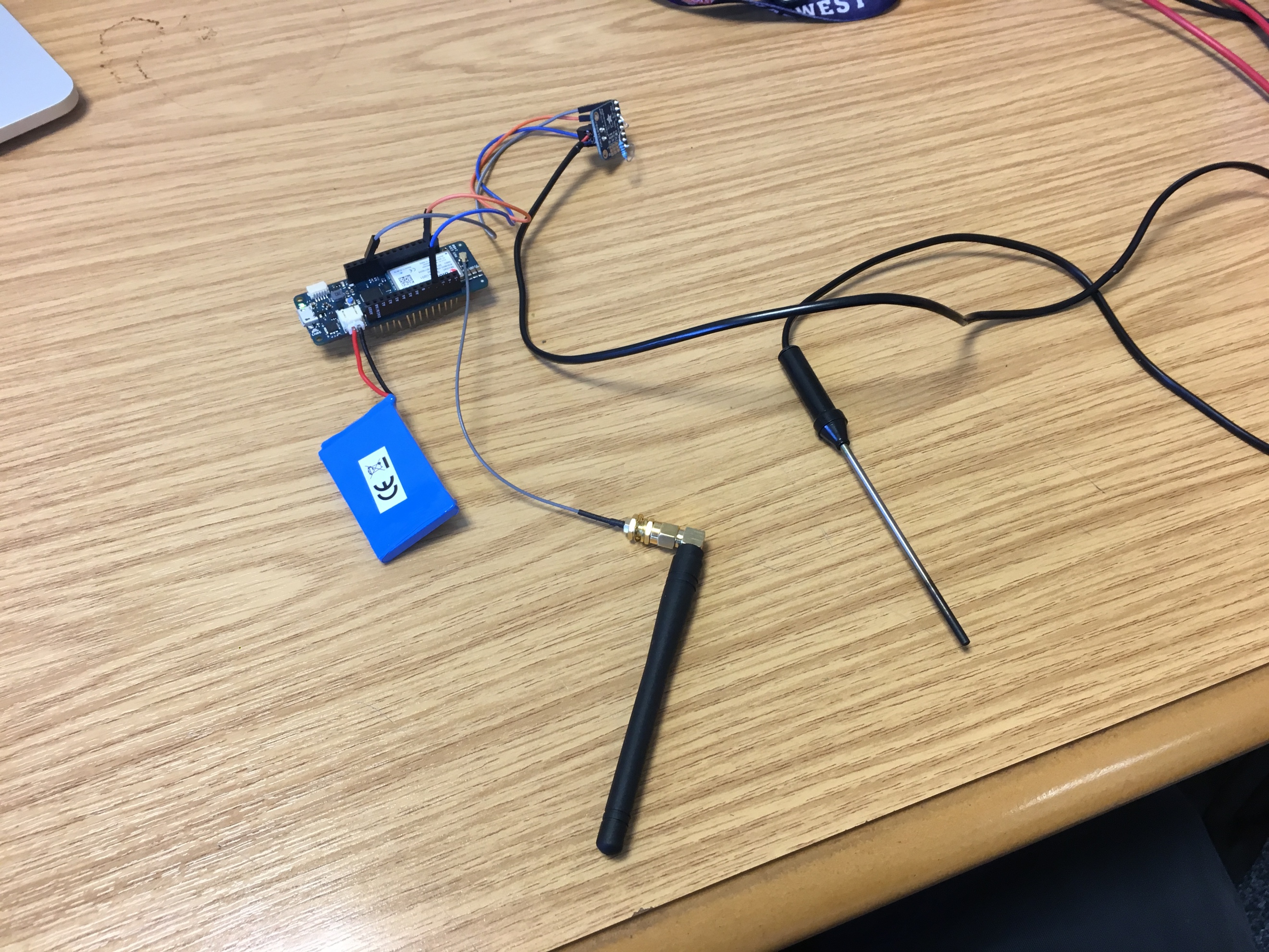


Figure 1 Prototype system

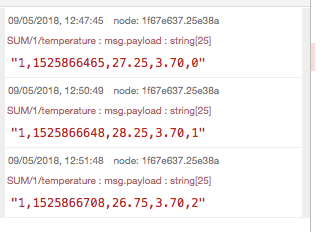


Figure 2 Data streaming in from Node-Red

**Low power optimisation:**

* SIP: The GSM modem is the largest energy consumer so SIP is used to reduce the number of transmissions
* Sensor switching: To reduce power for sensing, the sensor is only switched on long enough to take a reading.
* GSM switching: The GSM module is only left on long enough to make a transmission
* Sleep: When not doing anything, the node is put into a low power sleep state, the node wakes every minute with an interrupt from the RTC.



Figure 3 Thermocouple power on/off

**Power consumption**

* The node is powered by a 3.7V li-lo battery; a range of capacities are available (up to around 10,000mAh).
* Power consumption (timing):
  + Idle: 14mA (35S during a send period; 59.8S otherwise)
  + Sensing and processing: 24.5mA (134ms)
  + Transmitting: 80-150mA (25 seconds)
* Limitations:
  + We need to make more detailed measurements, especially during the transmitting period where the current draw changes frequently---code needs to be written in LabVIEW to read from the multimeter (est. 1-week job)
  + Lower power sleep states should be possible see the discussion regarding the mkr1200: https://forum.arduino.cc/index.php?topic=499101.0. The k1200 can achieve <500uA idle. I’m not sure why this is not the case for the MKR1400. I’ve posted on the MKR1400forum to ask for advice / tricks. I’ve also asked Arduino support.

**Lifetime estimate**

* Assumptions
  + Only 5% packets are transmitted
  + 10,000mAh battery
* We should achieve around 16 days with this setup.
* If power was reduced to a similar region as the MKR1200, then we would achieve approx. 72days+ lifetime

**Hardware fritzing diagram:**

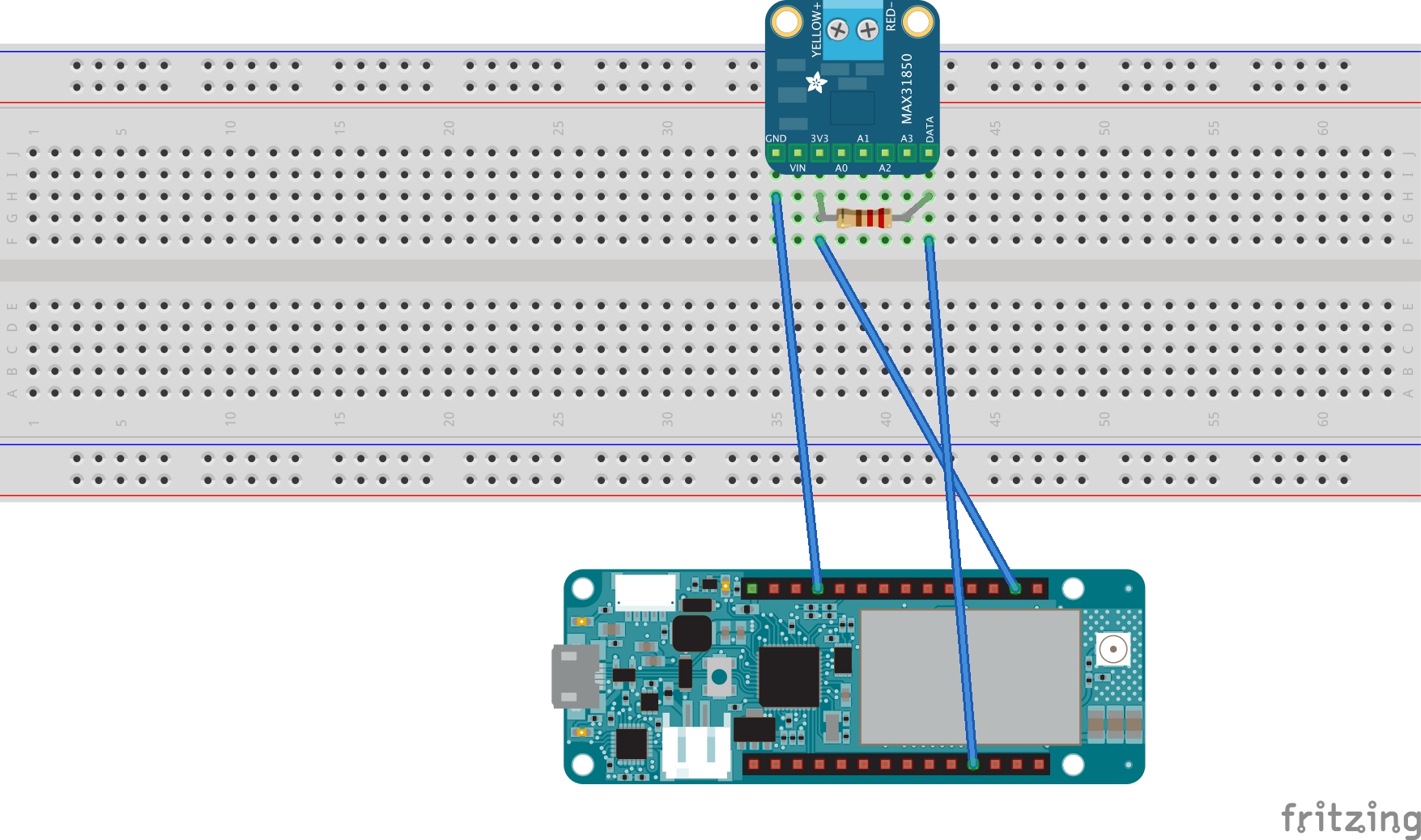


Figure 4 Fritzing diagram of MAX31850 connected to Arduino MKRGSM 1400

SVN Location: svn+ssh://cogentee.coventry.ac.uk/svn/HELP/SystemDevelopment/Software/Cookstove/SUM/Connections.fzz

**Code:**

SVN Location: svn+ssh://cogentee.coventry.ac.uk/svn/HELP/SystemDevelopment/Software/Cookstove/SUM

**Installation / running guide**

1. If setting up from scratch install MQTT on a remote server following the guide here: <https://diyprojects.io/mqtt-mosquitto-communicating-connected-objects-iot/#.WvGWMi-ZPs->
2. Install node-red to a laptop/server: <https://nodered.org/docs/getting-started/installation> and <https://www.digitalocean.com/community/tutorials/how-to-connect-your-internet-of-things-with-node-red-on-ubuntu-16-04> (step 3-5)
3. Checkout the code from SVN
4. In the svn folder copy node-red flow to flow directory (~/.node-red/lib/flows) then import and deploy (be sure to modify file location in node-red flow)
5. Connect up an Arduino following the hardware fritzing diagram (required components: MKR1400, MAX41850, k-type thermocouple, 4.7k resistor)
6. Open SUM.ino in the Arduino IDE
7. Modify SUM.h to set MQTT server address (line 22), SIP threshold (line 30), turn debug on/off (line 2). NOTE: Debug must be off when running from battery
8. Upload code to Arduino, power and run from a LiLo battery
9. Check data is being saved

**Long-term trial run**

* The node will be deployed in a conservatory for two-weeks from the 10th May 2018.
* The long-term test will test 1) the functionality of the node, 2) the battery life of the node.
* Live data will be viewable from: <http://159.65.25.153:1880/#flow/b2c9546.5c5f728>
* Note: With SIP being used, you may need to wait an hour or more to see data come in. (See Figure 2 as an example of what you should see).
* The data will be saved in the server’s home directory (“/home/ross/RossTest.csv”)

Things to do

CJC epoxy coating in enclosure

Packaging design

* 1. Battery change
  2. Portability
  3. Attachment to cookstove
  4. Fixture of thermocouple
  5. Fire resistance
  6. IP rating/suitability for environment

Battery size with heat

Battery voltage measurement

Lower power MKR1400

Test SIP with a cooking profile

Adaptive sample rate

Longevity test---simulated day---number of cooking events

Local data buffering in flash memory to reduce transmissions

Calibration

Power measurement

Field test

Purchasing for five systems

Purchasing Ibuttons

SUM: Robustness, data reliability, and longevity