About the code

There are three files:

- OTATelnet_Blynk_PCB_UA_4-31.ino, the main file.
- OTA_Telnet_2.h, for the communications components
- creds.h, the credentials file.

Main file

The settings for the triggers are gathered early on.

```
//// this section for the triggers >>>>>>>>>>>>>>>>>>>>>>>>>.
const float ramp = 4.0; // a sudden rise in temp will start the pump. Degrees C.
float minTemp = 45.0; // for getting the minimum of an array of temp values. Set it high to start. ramp will compare with this.
const float maxTemp = 50.0; // startup temp
const int gasTrigr = 4000; // this value will set off the system
const int gasTrigrLowr = 2500; // this value will set off the system in combination with elevated temp.
```

These constants allow you to vary the number of hose-lines you will be pumping, and the duration of water delivery for each hose-line.

```
const byte numLines = 7; // the number of water delivery pumping lines connected to the relays
const byte lineTime = 60; // how long each water line will pump for in seconds
```

>>>

The lineTime constant will work between 10 seconds and 255 seconds. For testing, I suggest 10 seconds. For practical use, 60 seems right for my house. If your local fire weather warning is for 20 plus C but high winds with very dry vegetation, plus existing fires in the area, you might lower minTemp and maxTemp. I set them high initially to avoid false starts and wasting scarce water.

If your situation requires only, say, four water lines, change the constant numLines appropriately.

```
const int c = 35; // y=mx+c equation for temp sensor calibration
const float m = 14.6; // y=mx+c equation for temp sensor calibration
```

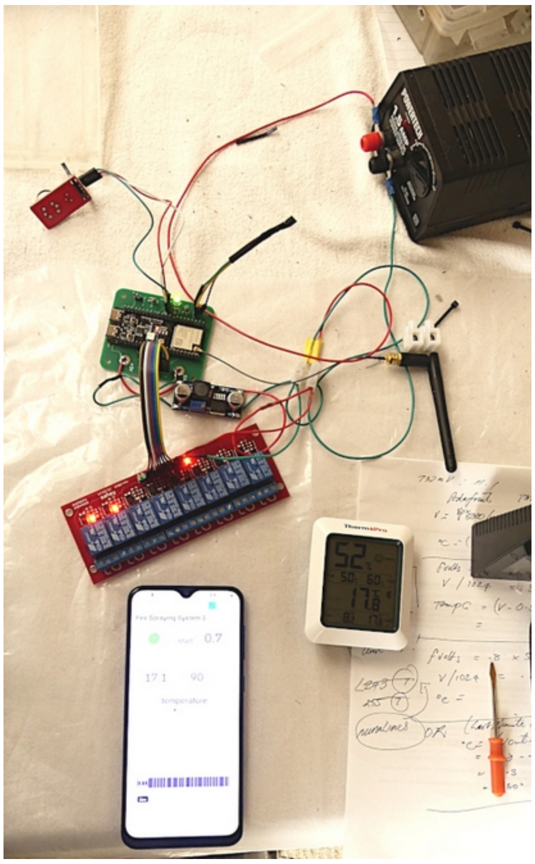
I have tried various temperature sensors and, in spite of the claims for some, they need calibration. The TMP36 sensor response is linear, although I have seen some log equations used. I set up a spreadsheet to plug in the voltage readings (float TMP36_mV in the code) and the temperature readings from a Thermopro TP350 temperature and humidity unit which should be accurate to 0.5 degrees C. A couple of coordinates are enough to calculate the equation. While this isn’t laboratory accurate it’s much closer than the specs settings. I have used 500 mV offset, from the specifications sheet. Next time I will connect the TMP36 to the 3.3v pin of the ESP-32, rather than the 5v I have now. My readings should then come in line with the specifications.

The function for temperature reading and averaging is

```
void checkSensorsAve() {
```

```
for (int j = 1; j <= numTempReads && analogRead(tempPin) > 0; j++) {
thisTempRead = (analogRead(tempPin));
tempReadBucket = tempReadBucket + thisTempRead;
}
```

numTempReads is set to 10. I have found that the sensor is capable of giving 500 or more readings each second, which one would think should give a better average. However the more reads, the more dropped reads, particularly if the unit is busy trying to connect to its communication sub-systems. Ten reads only each second gives a very stable result and leaves the remainder of the second for other processing activities. analogRead(tempPin) > 0 requires that a read can't be empty – which would upset the average.



Testing during development:

- checking the pumping duration and sequence
- checking the Blynk information was working and the start button functioned
- checking temperature calibration

I found the processor seems to be most stressed when the internet is patchy – quite likely in fire conditions. A number of programmers have suggested restarting the processor to give the best chance of reconnection. For this reason I built a counter to measure wifi downtime, and after 20 minutes (1200 seconds), the system restarts.

```
void restart() {
if ((downDecades > 120) && (started == 0)) ESP.restart();
}
```

The disadvantage of a restart is the loss of pumping cycle information. It's very useful to know how many times the unit has been triggered.

The pumping function, void pumping(), works through activating the relays for the pump and the water lines. The water line opens two seconds before the pump starts, so as not to stress the lines. The pump stops two seconds before the line closes. Then the next line sequence begins.

The float lineNowPumping is for checking where the pumping sequence is up to.
The float numPumpCycles shows the completed cycles.
Since I was limited with the number of variables I could use in Blynk, I combined these into what looks like a decimal number but isn't. On the phone app, Cycles/line of, say, 2.7, indicates there has been two completed cycles and the unit is pumping on line 7. When that last line is complete the number clicks over to 3.

creds.h

The credentials file isn't included because it's confidential. That's the idea.
Yours should look like this (keep the quotation marks):

```
const char* MYssid = "YourWiFiName";
const char* MYpassword = "YourWiFiPassword";

// Blynk
#define BLYNK_TEMPLATE_ID "YourBlynkTemplateXXXXXYYYYZZZ"
#define BLYNK_TEMPLATE_NAME "Fire spray system"
#define BLYNK_AUTH_TOKEN "YourBlynkToken"
```

>>>

GDM Tasmania 2024.

Part 2 – Software and Communication

Arduino to PCB – a steep learning curve

A number of people I have talked to expressed interest in the system, but were daunted by having to learn electronics and programming. This section explains a little of what is involved - those already conversant with Arduino programming, OTA, circuit design and board making etc may skip this section.

If you don’t like the idea of soldering and programming, you might find someone in your family or community who is willing to take this on. The plumbing of the hose lines is something most people can do.

The internet has great resources for training videos and discussion. The Arduino and Blynk websites have very useful introductions. Jaycar and other product supplier websites have not only Arduino compatible products, but also specification sheets explaining how to wire up the item and use it with a micro-controller. The Arduino IDE makes programming very straight-forward.

Arduino

As of February 2020, the Arduino community included about 30 million active users based on the IDE downloads. – https://en.wikipedia.org/wiki/Arduino

Arduino micro-controllers have been available for nearly 20 years and their programming and utility is very well developed. They have a reputation for reliability. The ‘integrated development environment’ handles the programming aspect, with a programming language check, upload to the Arduino board facility and a comprehensive library of ready to run software called ‘sketches’ – and much more.

For those interested in the challenge of learning to build projects and program the Arduino, or compatible devices of other brands, there are excellent learning resources online, including beginners courses. Paul McWhorter’s Arduino Lessons (https://toptechboy.com/arduino-lessons/) are comprehensive and entertaining. Although I must admit I’m still a novice programmer.

I was given an Arduino Uno to ‘start playing with’; thanks Gene. I blew it up after a few weeks by powering it from an external source, then, wanting to upload a changed sketch, I plugged the board into my computer in the usual way. Lights out. Not good. That encouraged me to work through the tutorials more carefully.

I soon realised it would be easier if I could upload the sketches wirelessly. This led to OTA or Over the Air programming – a great improvement. OTA however means you can’t use the serial port in the IDE to track and debug your sketch, so you need an external window, like Telnet. Fortunately someone had already written the code, it was a matter of using the right Google search to find it.

Then on to Blynk and the phone app. Once again there were many examples to look at, and simple sketches to begin with. Blynk also led to the choice of the ESP32 S3 board with an external antenna. I had found, for my situation, the low powered boards need this extra hardware.

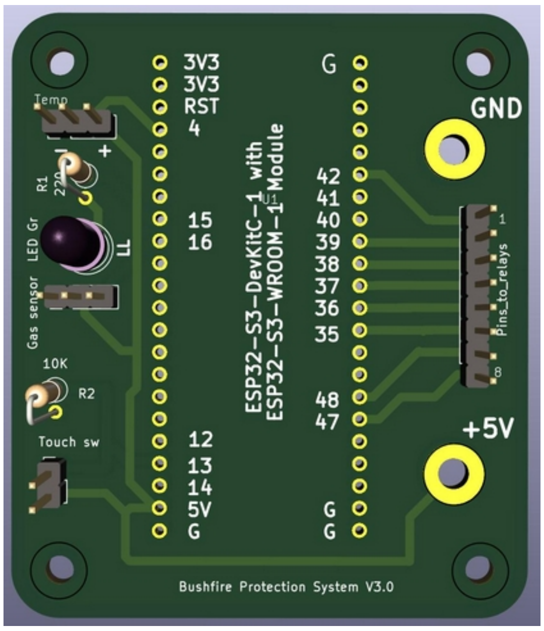
My program incorporates not only the triggers and pump start-stop relays, but also WiFi and OTA, Telnet and Blynk. And I had to make sure that the basic triggers would keep working if the WiFi was unavailable – quite possible in a real bush fire.

Pete Knight, a moderator on the Blynk forum, recommends ESP32 boards, and also, ‘keep your main loop clean’. The forum proved invaluable. The programming was more of a challenge than soldering up the parts.

Breadboards to PCBs

I started with a basic ‘push in the leads’ breadboard to test the idea. Wires everywhere! And hardly robust – after a week some of the connections were loose. On to a solderable breadboard with components on the front and fairly linear wiring on the reverse. Still rather messy and a dry joint was very hard to track down. The soldering was tricky too, I needed magnifiers to see what I was doing and a new soldering iron with a tiny point. I finally landed on a battery powered 6W soldering iron from Jaycar
https://www.jaycar.com.au/battery-powered-6w-soldering-iron/p/TS1535?srsltid=AfmBOooegESU0W7gDC9mIWZrKJpWLnCHWekQW4WTpALk9IwkM9n9iz5e
Only 6 watts? You couldn’t burn your finger on that could you? It’s very adequate for PCBs and has the added benefit of longer protection of the tip. You only heat it up when you need it (10 seconds). No cord! I bought some heavy duty re-chargeable batteries and I’m still on the first charge.

KiCad was suggested for PCB design. Thanks Gene.


Temp
R1
220
LED Gr
Gas sensor
10K
R2
Touch sw
3V3
3V3
RST
4
15
16
12
13
14
5V
G
ESP32-S3-DevKitC-1 with
ESP32-S3-WROOM-1 Module
G
42
41
40
39
38
37
36
35
48
47
G
G
GND
+5V
1
Pins_to_relays
8
Bushfire Protection System V3.0