# mqtt-mesh and mqtt-sound

Introducing two new display routines for Arduino and FastLED.

**mqtt-mesh** – Introduces the ability to synchronize and delay display sequences across multiple Arduino microcontrollers via IR control.

**mqtt-sound** – Adds sound reactive capability and display sequences to the notamesh framework. In this case, the delay is only the timing of sequence changes, and not with the response time of sound reactivity.

**Q. What is this ‘mesh’ all about?**

You can configure multiple Arduino microcontrollers with the same display sequence, but with a short delay when changing to a new routine, thus giving the effect of synchronization.

**Scenario:**

You have 5 IR controlled Arduino based displays, all within eyesight of each other, and all running mqtt-mesh. At compile time, each will be configured for the type of LED’s in use as well as a unique ID for each and with demo mode enabled.

Upon first time power-up, each strand will be configured with the length of the strand, as well as a delay. For instance, each Arduino has 40 LED’s connected to it.

Arduino A1 – 0ms delay

Arduino A2 – 100ms delay

Arduino A3 – 200ms delay

and so on . . . .

Once they’re all programmed and running, you would then press the ‘Reset’ button on the IR controller in order to reset the Arduino’s and synchronize millis(). The Arduinos would begin displaying the demo mode, with each Arduino slightly out of sync with the others.

**These sketches:**

* Run on an ESP8266.
* Use the FastLED display library.
* Use the pubsubclient library for MQTT messaging.
* Include a LOT of display routines.

My LED program animation design philosophy is to:

* NOT use delay statements in the loop as that breaks input routines (i.e. button).
* Not use nested loops (for performance reasons).
* Use millis() as a real time counter.
* Spend a bit more time on high school math, rather than just count pixels.
* Keep the display routines as short as possible.
* Data typing as strict as possible i.e. why define an **int** when a **uint8\_t** is all that is required. Floats are not used at all.
* Localize variables to each routine as much as possible, unless they are shared names that can be used for multiple routines.
* Break out the display routines into separate .h files for increased readability and modularity.
* Be generous with comments.

**Display Controls**

* Adjust overall brightness.
* Reset (to sync multiple controllers).
* Enable/disable demo mode.
* Increase/decrease speed of animation.
* Adjust overall brightness.
* Change direction of some routines.
* Enable/disable glitter.
* Select previous/next display routine.
* Enable/disable palette rotation.
* Select previous/next palette.
* Select lantern mode.
* Select lantern hue/saturation.
* Select hue/saturation based palette.

**EEPROM Functionality**

* Adjust strand length.
* Adjust mesh delay.
* Toggle direction of some displays.
* Change overall brightness.
* Toggle glitter.
* Change startup display mode.
* Change peak detection sensitivity (mqtt-sound only)
* Change noise filtering/squelch (mqtt-sound only).

**Sound functionality**

* High speed sampling of sounds via ADMP401 microphone.
* Signal smoothing.
* Averaging of last 32 samples.
* Improved peak detection.
* Integration of sound with various display routines.
* No MSGEQ7 required.
* Can replace microphone with a line-in signal on A0.