PART 3 – Recap & consolidate

Now we have the very basic elements of hardware working and talking to each other – host (server) node and a display node – I should review and make sure the basic plan is living up to expectations with what I want and have learned…

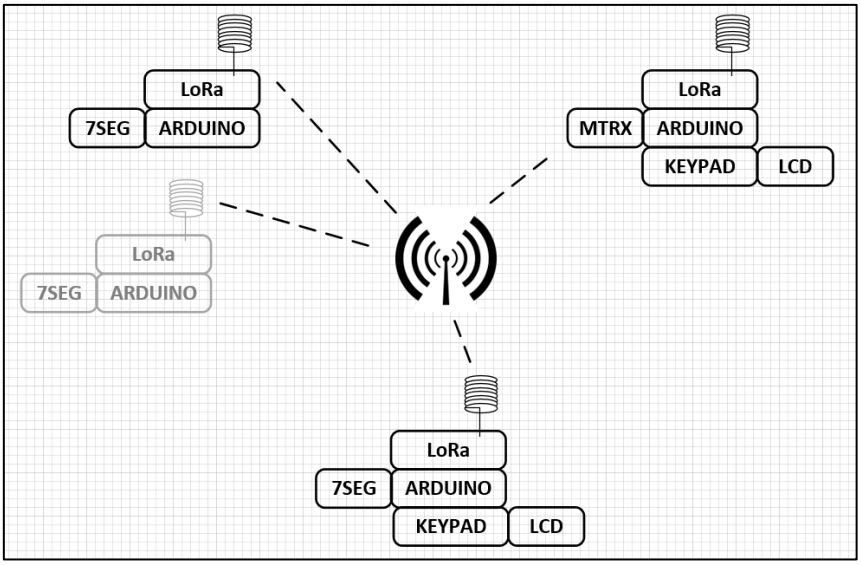
OPERATION

The 7-segment display node must send a LoRa 'request' asynchronously to the host for specific information on a regular basis. (Remember they initially have no awareness of each other until the radio nodes talk to each other.)

The host node needs to support a local user-interface, manage the client radio sessions, and maintain a data set for the overall application – supporting the fundamental CRUD (Create, Retrieve, Update & Delete operations on the host data.

This took a while, as the small memory model, and complexity of the related information needed some thinking. The data uses an array of static data in EEPROM (read-only in normal operation - to save EEPROM life), some persistent data for each node personality, and other RAM optimised arrays for run-time activities.

Similarly – there was a need to develop compact protocols for the LoRa radio links. Trying to be consistent across nodes will simplify the effort to revise and expand the 'network'



The whole project will involve four or more LoRa connected nodes that interact in real-time.

I chose to use the RadioHead LoRa library using the ReliableDatagram protocol – to ensure messages are sent & received.

LET'S SEE WHERE WE ARE FOR DATA REQUIREMENTS

As explained above – there are a few things happening – yet we are limited by mempory sizes – so the data nends to be compact and specific.

The NODE PERSONALITY info is stored in EEPROM, and read at start-up of each node

```
struct     node_type     { char       items_of_interest[6];    // the node will request these 'items'
                           unsigned int update_interval = 2000; // every xxx millisecs
                         };
```

SERVICE ITEMS are stored in EEPROM, read-only at runtime

```
struct     item_type     {   char name[ITEM_NAME_LENGTH];       // friendly name of each item
                             unsigned char units_per_serve;     // how many pieces sold per order
                             float price_per_serve;             // the sell price of each serve
                         };
```

ORDERS are manipulated in RAM as needed when each order is created or served.

```
struct     order_type     { unsigned int order_number;    // each order has a unique incrementing number
                          };
```

Each order may have multiple service items associated… (in RAM)

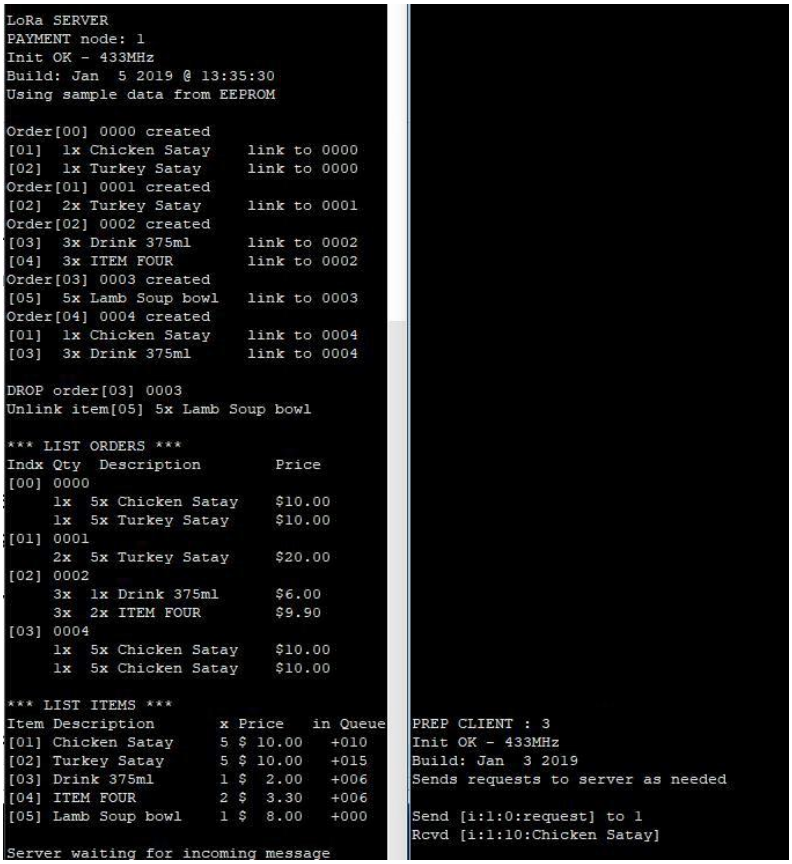
```
struct     order_item_type  { unsigned char order_index;   // reference to the 'order' number
                              unsigned char item_index;  // which item is attached to the order
                              unsigned char num_serves;   // how many of this item on this order
                            };
```

There are several other variables and arrays for transient data that isn't needed across resets.
HOWEVER – at the moment – if power is lost – the order stack and associated order_items are lost… this was a choice gainst available EEPROM and write-cycles. Perhaps a different non-volatile RAM could be added later.

This may change, but is a starting point for how we'll format the LoRa radio messages…

```
struct    msg_type       { char    type;           // what' the message 'about' (item, order… etc)
                           byte    index;          // refers to object[index]
                           int     qty;            // how many are involved ?
                           char    string[ITEM_NAME_LENGTH];   // message string if needed
                         };
```

This is where we are – showing the serial console outputs
- with the CLIENT also showing the relevant data on the seven segment LED display…



LoRa 'SERVER' initialising, and CLIENT node 3 waking up.

(Because there is no data-entry capability yet - sample data on the server node is pre-loaded)

So it looks like the basic elements are coming together for node and data activities.