Duck Packet Structure

```
/*
|0       |8       |16  |20|21|22|23  |27                                  255|
|        |        |    |  |  |  |    |                                       |
+--------+--------+----+--+--+--+----+---------------------------------------+
| SDUID  | DDUID  |MUID|T |DT|HC|DCRC|                DATA                   |
|        |        |    |  |  |  |    |           (max 229 bytes)             |
+--------+--------+----+--+--+--+----+---------------------------------------+

SDUID:     08  byte array          - Source Device Unique ID
DDUID:     08  byte array          - Destination Device Unique ID
MUID:      04  byte array          - Message unique ID
T   :      01  byte value          - Topic (topic 0..15 are reserved for internal use)
DT  :      01  byte value          - Duck Type
HC  :      01  byte value          - Hop count (the number of times the packet was relayed)
DCRC:      04  byte value          - Data section CRC
DATA:      229 byte array          - Data payload (e.g sensor read, text,...)
*/
```

Figure 1. CDP Packet Structure with byte sizes and very brief descriptions of each sections

DUID Functions(Device Unique ID)

Unique ID:

```
pd.setDuckId(duckutils::convertStringToVector("PAPA0001"));
```

Figure 2. Setting Custom DUID example for papa duck object pd

A Duck can be configured with a custom DUID but it MUST be 8 bytes.

SDUID (Source Device Unique ID)

SDUID is used to identify the original transmitting device for a message.

DDUID (Destination Device Unique ID)

DDUID is the destination of the message for CDP. It can be a specific duck or a group of them as shown below.

DDUID for Mass Transmission:

```
/**
 * @brief Use this DUID to send to all PapaDucks
 *
 */
static std::vector<byte> ZERO_DUID = {0x00, 0x00, 0x00, 0x00,
                                      0x00, 0x00, 0x00, 0x00};

/**
 * @brief Use this DUID to be received by every duck in the network
 *
 */
static std::vector<byte> BROADCAST_DUID = {0xFF, 0xFF, 0xFF, 0xFF,
                                           0xFF, 0xFF, 0xFF, 0xFF};

/**
 * @brief Use this DUID to be received by every duck in the network
 *
 */
static std::vector<byte> PAPADUCK_DUID = {0x50, 0x61, 0x70, 0x61,
                                          0x44, 0x75, 0x63, 0x6B};
```

Figure 3. Default DUIDs for mass transmission.

Undefined DUIDs:

```
std::string createUuid(int length) {
  std::string msg = "";
  int i;

  for (i = 0; i < length; i++) {
    byte randomValue = random(36);
    if (randomValue < 26) {
      msg = msg + char(randomValue + 'a');
    } else {
      msg = msg + char((randomValue - 26) + '0');
    }
  }
  return msg;
}
```

Figure 4.Undefined UUIDs are assigned a random value. (Defaults to 8 bytes) Not implemented in Quad Pro Prototype [From CDP github]

MUID (Message Unique ID)

```
void Packet::getMessageId(BloomFilter *filter, uint8_t
message_id[MUID_LENGTH])
{
  bool getNewUnique = true;
  while (getNewUnique)
  {
    duckutils::getRandomBytes(MUID_LENGTH, message_id);
    getNewUnique = filter->bloom_check(message_id, MUID_LENGTH);
    cout << "prepareForSending: new MUID -> " <<
duckutils::convertToHex(message_id, MUID_LENGTH).c_str() << endl;
  }
}
```

Figure X: code showing how MUID is generated for the packet

An MUID is generated using random characters, hashed and then checks if that hash already exists in either of the 2 bloom filters and if it has been used. If it hasn't the MUID can be used and if it has been used it will generate another MUID until the MUID hasn't been used.

T (Topic)

Topics identify the type of data transmitted from the source to the destination. You can see below what data is there.

```
/**
 * @brief Defines preset topics for duck packets.
 *
 */
enum topics {
  /// generic message (e.g non emergency messages)
  status = 0x10,
  /// captive portal message
  cpm = 0x11,
  /// a gps or geo location (e.g longitude/latitude)
  location = 0x12,
  /// sensor captured data
  sensor = 0x13,
  /// an allert message that should be given immediate attention
  alert = 0x14,
  /// Device health status
  health = 0x15,
  // Send duck commands
  dcmd = 0x16,
  // MQ7 Gas Sensor
  mq7 = 0xEF,
  // GP2Y Dust Sensor
  gp2y = 0xFA,
  // bmp280
  bmp280 = 0xFB,
  // DHT11 sensor
  dht11 = 0xFC,
  // ir sensor
  pir = 0xFD,
  // bmp180
  bmp180 = 0xFE,
  // Max supported topics
  max_topics = 0xFF
};
```

Figure 6. Preset Topics for default Duck Functions.

```
enum reservedTopic {
  unused = 0x00,
  ping = 0x01,
  pong = 0x02,
  gps = 0x03,
  ack = 0x04,
  cmd = 0x05,
  max_reserved = 0x0F
};
```

Figure 7. Reserved topics for Duck Packets.

DT (Duck Type)

Identifies the type of duck a transmission originates from:

```
/**
 * @brief Type of ducks
 *
 */
enum DuckType {
  /// A Duck of unknown type
  UNKNOWN = 0x00,
  /// A PapaDuck
  PAPA = 0x01,
  /// A MamaDuck
  MAMA = 0x02,
  /// A DuckLink
  LINK = 0x03,
  /// A Detector Duck
  DETECTOR = 0x04,
  MAX_TYPE
};
```

Figure 10. Enumerated Duck Types. [DuckTypes.h from CDP github]

HC (Hop Count)

Number of times a packet is transmitted/ retransmitted in the mesh network. A packet is only retransmitted if it is being seen for the first time by the Mama duck.

DCRC (Data Cyclic Redundancy check)

The cyclic redundancy check (CRC) is used to verify data integrity. The CRC calculates the checksum of the data section. It can be used to verify if the received packets data has been corrupted or not by comparing the CRCs between what was in the packet and what was calculated from the received Data.

DATA

The Data for the specific topic. (For example a gps NEMA string with latitude/longtitude, text message, etc)

Topics That affect Data format

ACK

```
/*
Data Section of a broadcast ack (max 229 bytes):
0 1      8    12...
| |      |    |
+-+------+----+-...
|N| DUID |MUID|
+-+------+----+-...

Maximum number of DUID/MUID pairs is:
  floor( (maximum data payload size - N) / (DUID size + MUID size) )
Which is:
  floor( (229 - 1) / 12 ) = 19

N:              1  byte               - Number of DUID/MUID pairs
Each DUID:     08  byte array         - A Device Unique ID
Each MUID:     04  byte array         - Message unique ID
*/
#define MAX_MUID_PER_ACK 19
```

Figure 7. Construction of a Broadcast Ack Data Section

Ack (Acknowledgement) is transmitted from receiving duck to transmitting duck to confirm reception of messages/ packets

CMD

```
/*
Data Section of a duck command (max 229 bytes):
0 1    229...
| |     |
+-+-----+-...
|N| VAL
+-+-----+-...

N:             1  byte              - Command lookup value
VAL:         228  bytes             - Value to set for command

*/
```

Figure 8. Construction of a Duck Command Data Section. [CdpPacket.h]

Duck commands are listed

```
//Available command IDs (N)
#define CMD_HEALTH 0
#define CMD_WIFI 1
#define CMD_CHANNEL 2
```

Figure 9. Default Command IDs used to transmit the type of command.

HEALTH - Checks if the mama duck is in good condition by telling the mamaduck to send a health packet to all papaducks.

WIFI - Can turn on/ off WIFI connection If there is one. (This is not in the Quad Pro Prototype)

CHANNEL - modifies the frequencies of the radio(This is not in the Quad Pro Prototype)

How to Send/Receive

How send a packet:

```
/*-----setup duck link begin -------*/
dl.setDuckId(duckutils::convertStringToVector("DUCK0001"));

/*-----setup duck link end -------*/


dl.prepareForSending(&filter, dduid, topic, dl.getType(), 0x00, data);


/*---------Send to Lora begin---------*/
payload = dl.getBuffer();
cdppayload = duckutils::convertVectorToString(payload);

cdppayload = encodeCDP(cdppayload);

cout << cdppayload << endl;

cdppayload = decodeCDP(cdppayload);
cout << cdppayload;
 cdppayload = encodeCDP(cdppayload);
 cout << cdppayload << endl;
//cdppayload = modifystring(cdppayload, TOPIC_POS)

publish(redisConnect, redisConfigLora.stream_name, "CDP_LORA", cdppayload, redisConfigLora.response);
/*---------Send to Lora end---------*/
```

Figure X: how to create and send a CDP packet

How receive a packet and handle packet based on duck:

```
md.setDuckId(duckutils::convertStringToVector("MAMA0001"));

while(!messageReceivedLora){
    sleep(5);
    read_from_consumer_group(redisConnect, redisConfigLora.stream_name, redisConfigLora.group_name, redisConfigLora.consumer_name, redisConfigLora.rxKey, redisConfigLora.key_buffer,redisConfigLora.messageBuffer, redisConfigLora.messageID, redisConfigLora.txLoraQueue, redisConfigLora.task)
    messageReceivedLora = acknowledge_message(redisConnect, redisConfigLora.stream_name, redisConfigLora.group_name, redisConfigLora.messageID);

}
dequeue_task(redisConnect, redisConfigLora.rxLoraQueue, redisConfigLora.task);

cdppayload = decodeCDP(redisConfigLora.task);

dp.setBuffer(duckutils::convertStringToVector(cdppayload));

//sends the redis messages in function below
md.handleReceivedPacket(dp);
```

Figure X: mama duck steps to handle packets

```
pd.setDuckId(duckutils::convertStringToVector("PAPA0001"));

/*-------Read from lora begin-------------*/
while(!messageReceivedLora){
    sleep(5);
    read_from_consumer_group(redisConnect, redisConfigLora.stream_name, redisConfigLora.group_name, redisConfigLora.consumer_name, redisConfigLora.rxKey, redisConfigLora.key_buffer,redisConfigLora.messageBuffer, redisConfigLora.messageID, redisConfigLora.txLoraQueue, redisConfigLora.task)
    messageReceivedLora = acknowledge_message(redisConnect, redisConfigLora.stream_name, redisConfigLora.group_name, redisConfigLora.messageID);

}
dequeue_task(redisConnect, redisConfigLora.rxLoraQueue, redisConfigLora.task);

cdppayload = decodeCDP(redisConfigLora.task);


dp.setBuffer(duckutils::convertStringToVector(cdppayload));
/*-------Read from lora end-------------*/


pd.handleReceivedPacket(dp);


/*-------Send to web server begin----------*/
receivedData = dp.getBuffer();

string messageForWeb = sendToWeb(receivedData);

publish(redisConnect, redisConfigWeb.stream_name, "CDP_WEB", messageForWeb, redisConfigWeb.response);

redisConfigLora.messageBuffer.clear();
```

Figure X: papa duck steps to handle packets

Detector Duck Functionality

Explanation:

This duck is a unique duck that can ping other ducks and is used to check how strong the signal is by checking its RSSI (Received Signal Strength Indicator). It is important to note that it can really only ping the mamaducks because papaducks ignore pings and ducklinks dont receive data. The current detector duck code needs to retrieve the RSSI directly from the LoRa radio but other than that this duck will work fine.

DUCK LINK Function Implementation

Explanation:

The duck link is the duck that only transmits data to the rest of the duck network. It first has to receive a message from the redis Stream published by the Web Server with the key “WEB_CDP” about the data and topic the user wants to send. From the information from the web server it will form the packet and send the data to the radio by publishing a message to the redis stream with the key “CDP_LORA” to transmit the data.

More details about packet handling:

- The CDP packet data when it is initially generated is a vector of bytes. This is important because the formulation of the packet has characters that are guaranteed won't go into the string properly and leave symbols that can be misunderstood when converted back into a vector (This is a link to the ascii table that tells you what numbers are what character: ASCII table - Table of ASCII codes, characters and symbols (ascii-code.com).

- After the packet is generated the Topic section all the way through the DCRC section of the packet is not going to be read properly as a string so It has to be modified so all of the data of the packet can be read and sent to the LoRa radio. This has already been taken care of and is undone to read the data in the current Quad Pro prototype.

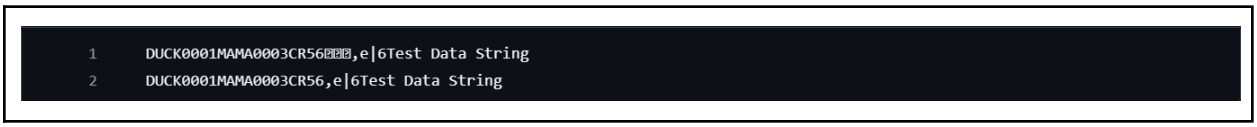

Figure X: The first string is an example of an unmodified string once CDP payload is generated. The string below is what will be read by the LoRa radio showing how important it is to make sure every part of the string is readable.

- The Broadcast DUID vector will also not be interpreted properly because each byte is bigger than the max value for the readable ascii range. One thing you can do to make it readable is after the CDP packet is a string to replace the DUID of the packet to “FFFFFFFF”. And to read the original packet of data you just replace the string “FFFFFFFF” to the broadcast DUID vector.

MAMA Duck Packet Functions

Explanation:

The received data comes from the LoRa radio using redis messages with the key “LORA_CDP”. The data will be converted back into its original data to relay CDP packets and communicate with the papa duck so the CDP packet gets sent to the WebServer. Depending on the topic the mama duck will create and send packets as described below back by publishing the data with the key “CDP_LORA” to transmit.

More details about packet handling:

- The bloom filter is used to ensure that the duck has not already seen (and relayed) the packet that has been received. If the duck hasn’t already retransmitted the packet, it adds the packet’s MUID to the bloom filter, increments the hop count, assigns updates to the rx packet byte buffer.

- The mama duck sends acknowledgements to papa ducks that messages were transmitted successfully by checking its device ID and the last MUID used in the bloom filter of the mamaduck. If it's true it will send the acknowledgement to the papaduck

- Only the mamaduck handles commands and duck commands. It is important to note that duck commands are different from commands. This distinction is important because duck commands don't do anything at the current moment according to the original CDP github made by project OWL.

- If a mamaduck receives a command it will send an acknowledgement to all papaducks. This ack is different from the papaduck acknowledgement buffer because it is only sending the mamaducks SDUID and the MUID generated for this packet.

PAPA Duck Packet Functions

Explanation:

The Papa Duck will take data from the LoRa radio by reading the messages from the redis stream with the key “LORA_CDP”, convert it back into its original contents and send it to the Web Server so you can see the message and send other messages back to other ducks. How it will do this is explained below.

More details about packet handling:

- Similar to the mama duck, The bloom filter for this duck is to ensure that the duck has not already seen (and relayed) the packet that has been received. If the duck hasn’t already retransmitted the packet, it adds the packet’s MUID to the bloom filter,

- The papa duck does ignore pings and delete the packet received if it is a ping

- If acknowledgements are enabled, it will check if the CDP packet needs an acknowledgment first and request the acknowledgements from the mama duck by broadcasting an ack request. If it receives an Ack it will send the data to the web server although this is in the quad pro prototype this feature has been ignored at the moment.

References

1. Duck packet structure along with some code references used.
   ClusterDuck-Protocol/ClusterDuck-Protocol: Firmware for an ad-hoc mesh network of Internet-of-Things devices based on LoRa (Long Range radio) that can be deployed quickly and at low cost. (github.com)
2. What is a bloom filter?
   Bloom Filters - Introduction and Implementation - GeeksforGeeks
3. Ascii Table. Important to know for the data that can interpreted properly for a string
   ASCII table - Table of ASCII codes, characters and symbols (ascii-code.com)