1. Basic concept or idea – creating a portable mesh network for S&R
2. Find a product that looks usable, has what looks like decent number of recourses and has some repositories to use and roll with it.
3. Figure out how they work and how to set them up.
4. Figure out how to get them to communicate
   1. ended up testing in transparent mode and then API mode.
5. Get one radio to transmit anything from a teensy to a receiving unit.
   1. Able to get a teensy to count and send a signal in transparent mode every second or two to a receiver and have it print out the data. As I was able to prove that tx was possible, I did a small distance check around the school to see how far in line of sight and what would block the signal.
6. Get teensy to send an API packet to receiver and have it print out data.
   1. Created a packet in XCTU program, put that into a buffer to send. Ended up manually adding “0x##, ” to all values in buffer to get it to send.
   2. Now needed to get specific data in the packet – had to set up GPS.
7. Set up GPS on teensy and figured out basic setting to see data (repository was good and worked with minimal troubleshooting)
8. Set up OLED so I can print data to it and see what’s being printed out.
9. Getting values from the GPS, put those into the XCTU API frame generator.
   1. Manually created a new packet in teensy
10. Verified packet could be received with the specific data.
11. Set up Argon to receive data.
    1. Tried to use repository but could not get it to work.
    2. Ended up using a read if available, print data to buffer

if(Serial1.available()>0){

    tempData[i] = Serial1.read();

    Serial.printf(" %i 0x%X, ", i, tempData[i]);

  i++;

1. An issue with incoming data
   1. Needed to put in a reset for the increment. If the increment went too far over the buffer, particle would reset and then increment would go back to 0.
2. Printed out buffer to make sure all data was copied over.
   1. Comparing to the original packet, found out of the data was missing or changed.
   2. Upon further inspection this is part of the addressing or packeting information before the actual data. *The main data section was unchanged.* Most of the information before and after was different.
3. After finding this out, needed to figure out a way to isolate just the data
   1. The issue would be that the packet will be a different size than the original and that the data can vary both potentially in size and location.
   2. Thought about removing excessive into such as removing the decimal. Found a function that prints just numbers with no decimal. “latitude\_fixed” // fixed lat is 350844096 // fixed long is -1066463296.
4. Delineate latitude and longitude.
   1. Chose pipes for delineating lat and long from the rest of data. //buf is |350844096|-1066463296|.
5. Search for these specific characters and strip data from there
   1. Xbee library did not work – manual search and strip
   2. How do you search for a specific character and extract the data?

void findPipe(){

  const char P\_LEN = 5;

  char pipe[P\_LEN];

  uint8\_t i,p, latSize, lonSize;

  char lat[15], lon[15];

  p=0;

  // Serial.printf("Chekcing for data");

  if (tempData[0] != 0x7E){

    Serial.printf("No Data, leaving function\n");

    return;

  }

  Serial.printf("Start finding pipes");

  for (i=0; i<BUF\_LEN; i++){

   if (tempData[i] == 0x7C){

      pipe[p] = i;

      Serial.printf("| at tempData[%i], pipe[%i] = %i\n", i, p, i);

      p++;

      }

    }

  latSize = pipe[1]-pipe[0]-1;

  lonSize = pipe[2]-pipe[1]-1;

  Serial.printf("latSize = %i lonSize = %i\n", latSize, lonSize);

  memcpy(lat, &tempData[pipe[0]+1], latSize);

  memcpy(lon, &tempData[pipe[1]+1], lonSize);

  actualLat = atoi((char \*)lat)/10000000.0;

  actualLon = atoi((char \*)lon)/10000000.0;

  Serial.printf("Actual Lat = %f\n Actual Lon = %f\n", actualLat, actualLon);

1. After extracting the data and creating floats, now we need to use live data.
   1. At this point we need to make sure we’re able to create API packets with live GPS data.
   2. Found a library that had a partially written out packet create and check sum.
   3. Following their setup as a guide and referencing the API data sheet for xbee, I found a few issues. I was having some Checksum issues due to Part of their creating the packet setup.
   4. “txBuf[2] = sizeof(txBuf) + 14;” //LSB - Packet created with no additional data show 0E in XCTU so needs size of () + 14.

Graphical user interface, application

Description automatically generated

* 1. Double checking their input looked correct, but I was still getting bad checksum numbers and the receiving unit wouldn’t accept the packet.
  2. Putting the new buffer into the XCTU software and seeing what their checksum number was compared to mine, it was a difference of 18.
  3. Removing the 14 to that specific part brought down the difference to 4. The new input for txBuf[2] would be sizeof(txBuf) – 4.
  4. Checksum is now correct and receiver accepts packets.

1. turn them into JSON buffers to send to a dashboard
2. Create Adafruit IO feed and dashboard.
3. Setup connection
4. Verify data sends and shows up in feed and dashboard
5. Remove button and auto send to dashboard
   1. After removing the button, an issue of how the data was being received and put into a buffer come up.
   2. Setting as a while statement and setting “i=0” after still set the buffer to always use position [0].

A screenshot of a computer

Description automatically generated

* 1. Setting it as an if else statement ended up working. Apparently available cycles between 1 and 0 for each segment of data.

A screenshot of a computer

Description automatically generated

* 1. Now we need to have a way for it to auto extract and send, but only there’s actual data that’s come in.

  if(Serial1.available() == 0 && millis()-timerDelay >= 500){

    i=0;

    findPipe();

  }

* 1. Adding in the 500 millis() timer allows the cycle that happens when data comes in to not accidently be cut off early. A packet on average takes 500 millis() for all the data to be received and printed.

1. Now that it auto runs findPipe until data come in, we now have a flooding issue for the Dashboard. Putting in a check for the pack delimiter 7E or ASCII “ ~ “, we are able to prevent the function from continuing and flooding.

  // Serial.printf("Chekcing for data");

  if (tempData[0] != 0x7E){

    // Serial.printf("No Data, leaving function\n");

    return;

  }

1. With everything working as expected one last issue came up. If the MQTT protocol doesn’t send/receive data after around 2 minutes, it will close the connection.
   1. Added a pinging function to ensure this does not happen.
2. The program works as intended, only sending GPS data to a dashboard when it received a verified pack.