LoRa-Based Image Transmission System: Algorithm Documentation

I. Introduction

This document provides a detailed explanation of the algorithms implemented in the LoRa-based image transmission system. The system consists of a LoRa node and a laptop server: - **LoRa Node**: Built using a Raspberry Pi Zero 2W, a LoRa pHAT from RAK, and a USB camera. The node captures images, processes pixel data for optimization, and transmits the data to The Things Network (TTN) via the LoRaWAN protocol. - **Laptop Server**: Receives image data forwarded by TTN using a webhook integration powered by ngrok. It processes binary chunks and reconstructs the image dynamically.

—

II. LoRa Node Algorithm

The LoRa node continuously captures and transmits images in an efficient format. This process reduces data size by optimizing pixel data and transmitting only changed pixels after the first image.

—

III. Laptop Algorithm

The laptop receives image data from TTN via a webhook and reconstructs the images dynamically. This process involves decoding binary data, updating the image pixel-by-pixel, and saving the result.

—

IV. Conclusion

This document outlines the detailed workflow of the LoRa node and laptop systems used for efficient image transmission and reconstruction. The algorithms are optimized for minimal data transfer while maintaining accuracy, leveraging TTN for cloud routing and ngrok for webhook integration. This system highlights the potential of LoRaWAN in multimedia IoT applications.

Algorithm 1 LoRa_Node_ImageTransmit

Inputs: - Camera for image capture. - LoRa pHAT for communication. - Chunk size for data division.
Outputs: Optimized image data sent to TTN.

Initialize System:
- Configure the LoRa pHAT to OTAA mode and set the frequency region to US915.
- Join the LoRaWAN network using TTN.

while True **do** ▷ Infinite loop for continuous operation

Capture Image:
- Use the USB camera to capture an image.
- Save the image in JPEG format and resize it to 128×128 pixels.

if Previous image exists **then**

Calculate Pixel Differences:
- Compare the current image with the previous image using pixel-by-pixel comparison.
- Identify changed pixels using the `ImageChops.difference()` function.
- Store changed pixels as (x_1, x_2, y, r, g, b).

else

Optimize Current Image:
- Traverse each row to identify horizontal runs of pixels with the same color.
- Represent each run as (x_1, x_2, y, r, g, b) to reduce data size.

end if

Divide Data into Chunks:
- Split optimized or difference data into chunks of size `chunk_size`.
- Add metadata (total number of chunks) to the first chunk.

Transmit Data:
- Send each chunk via the `lora.send()` method.
- Log transmission progress and handle any errors.

Update Reference Image:
- Save the current image as the ”previous image” for use in the next iteration.

end while

Algorithm 2 Laptop_ImageReceive

Inputs: - Data chunks received from TTN via webhook. - Metadata for total number of chunks.
Outputs: Reconstructed images saved incrementally.

Initialize Flask Server:
- Set up a directory for saving received images.
- Initialize a blank 128×128 RGB image as `reconstructed_image`.
- Set counters: `image_count`, `total_chunks`, and `processed_chunks`.

while Receiving Data **do**
  if First Chunk Received **then**
    Extract metadata to determine `total_chunks`.
    Decode remaining binary data.
  end if
  Process Chunk:
- Unpack binary data into segments (x_1, x_2, y, r, g, b).
- For each segment:
  - Update pixels from x_1 to x_2 in row y with the color (r, g, b).

  Increment `processed_chunks`.
  if All Chunks Processed **then**
    Save the reconstructed image as `reconstructed_image{image_count}.png`.
    Increment `image_count`.
    Reset `total_chunks` and `processed_chunks` for the next image.
  end if
end while