StormNet – a Low-Cost Communication Infrastructure for Disaster Scenarios

by Joshua Schmidt, Jared Kantor, and Lucas Payette

Abstract

As the Earth gets warmer due to climate change, natural disaster scenarios are increasingly larger and common. Firefighters, the red cross, and emergency workers are overburdened with these extreme weather events, especially when vital communication networks are damaged. In the event of a network outage, rescue workers are unable to leverage the power of the internet, and are forced to use inefficient methods of organization and communication. This project tackles these problems by establishing a low power communications network using a "mesh" of multiple wireless access points. These low-power nodes create a network over the disaster area, hosting a web-based communications platform providing limited messaging and organizational capabilities. With ESP32 SoC microcontrollers facilitating the network infrastructure, this solution enables a low-latency 433Mhz LoRa connection that can span 2.6 km per access point. We created a medium-bandwidth, battery-powered mesh-network, designed and built a disaster-relief web application, and analyzed the overall performance of this solution.

Introduction

Currently emergency workers use long-distance handheld radios for communication, with helicopters and drones used to survey the disaster site. However, it is difficult to communicate with the victims themselves, and it takes time to find civilians who are lost or trapped in debris. There needs to be an easier way for civilians to work with rescue workers, to give them vital information in an organized fashion so that resources can be allocated in the most efficient way possible. Since a cellular network cannot be relied on, and satellite connections are expensive, the best way to connect civilians to rescue workers is through a low-cost, battery-powered mesh network that is separate from the internet. The nodes in this network would be deployed with drones in populated areas after the natural disaster is over. These packages would contain some first-aid supplies, survival essentials, and the network node itself, powered by a lithium-ion battery pack. The entire package costs less than $15 US to produce, and allows for easy communication with emergency services in an intuitive manner.

Hardware

The network node is at the heart of the care package, which includes simple instructions for connecting to the network. The node is comprised of the ESP32 microcontroller with a LoRa radio and an oled display to allow for easy debugging. The microcontroller is powered by a simple 18650 lithium ion battery pack, and can optionally be connected to a gps module to keep track of the node's location. Each of these components is low-cost and easily sourced, making this solution very scalable.

Design

We wanted to make this system as easy to use as possible. The first thing we focused on was user interaction and user interface, putting ourselves in the shoes of a natural disaster survivor and simulating how they would use this communications system. Our final mockup can be viewed on figma here. We created four different pages - announcements and updates, map, message, and emergency call. In announcements survivors can see the progress that emergency workers make as they help people evacuate the area. On the map screen they can see where rescue workers are at any given time, as well as survivors using the application. The message screen and emergency call screens act just like a texting and calling app - very straight-forward. For the demo we decided to focus on the messaging portion of the app, as that was the medium that would be most beneficial to emergency responders.

Software

The software was by far the hardest part of this project. There were two sides, the web application and the code for the microcontroller, both of which had their own challenges. First of all, we decided to create a web application as opposed to a mobile app so that civilians did not have to download anything before the network goes down. Instead, they would connect to the wifi network provided by their node, and they would navigate to `stormnet.com`, which would serve a webpage hosted by the microcontroller itself. The ESP32 only has 8mb of flash memory – SPIFFS – and cannot serve files over 128kb in size easily. It was difficult to find a javascript framework that would produce static web pages small enough to be served by the microcontroller, but we eventually settled on Nuxt, a Vue.js-based framework for producing small static websites. On the microcontroller side, there were many challenges, ranging from sending small-enough packets to managing websocket connections to routing the packets from the source to the destination and back. We ended up creating our own packet type for sending over the network, complete with the payload, destination id, sender id, frame id, packet id, length, and message type. We chose not to use tcp or a different standard packet type because of bandwidth constraints from the long-distance radio. PlatformIO was used to manage the libraries used for the microcontroller, and the source code for the project can be found here. To improve upon the current software stack, we would focus on adding ssl support, and creating more intelligent routing of packets based on geographic location.

Conclusion

This project was largely successful, but there is still much that can be improved upon. SSL is a necessity, but we could not provide it due to the libraries that being used and lack of secure web socket support. Adding routing between connected websocket clients is the next major challenge, but this requires complex handshakes (a first attempt can be found on the master branch of the repo). Finally, more intelligent routing is needed in order to make this solution truly scalable, and this routing should be focused on sending the packet in the shortest physical path from sender to receiver. There are many areas that can be improved, and much more that can be added to this project, but overall the solution works fairly well.

links

- repo: https://github.com/jschmidtnj/cpe490
- mockup: https://www.figma.com/proto/j7aMdhLXvZLuetLyyZX5pt/Storm-Net