SafeDetect Product Documentation

1st Charles Andre
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
candre@usc.edu

2nd Yutong Gu
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
yutonggu@usc.edu

3rd Sharon Guo
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
guox@usc.edu

4th Sufyan Shaikh
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, USA
sufyansh@usc.edu

***Abstract*—SafeDetect is a product developed over the course of the University of Southern California's Internet and Cloud Computing course. By utilizing an audio detector, multitech xdot, raspberry pi, AWS and a machine learning model we are able to detect gunshots and triangulate their location. This paper will go over motivation, implementation, and results.**

***Index Terms*—AWS, xDot, LoRA, Multitech, Raspberry Pi, Lambda, MFCC, Machine Learning, Cloud, LoRAWAN**

I. Introduction

Gun violence has been steadily increasing over the past few decades [1]. When a person is shot they need urgent medical treatment in order to increase their chances of survival. In the absence of a gunshot detection system, this means that a good citizen must call in the gunshot if emergency services have any chance of helping the victim. This delay and reliance on human intervention can lead to a delay or complete absence of medical help. Almost any gunshot wound can become fatal if not treated quickly.

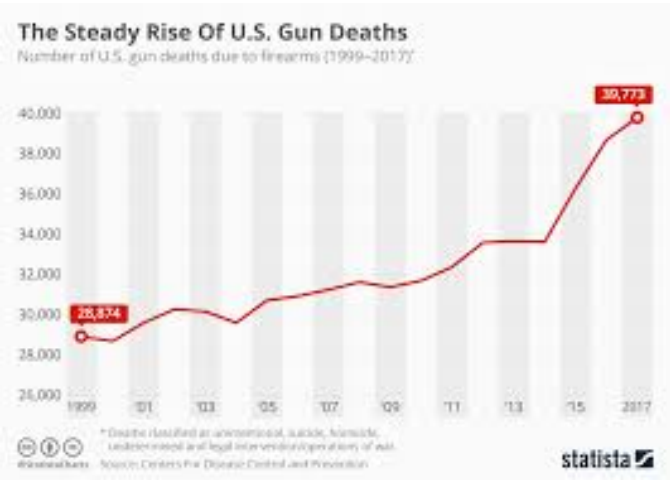

Fig. 1. Gun Deaths have been on the rise over the past 20 years.

The move for the future is audio based automatic gunshot detection [2]. In this paper we propose SafeDetect as a solution accurately detecting gunshots and alerting the police. By removing the need for human detection and reporting, our system will lead to faster response times. This will help get emergency response to gunshot victims as quickly as possible. Our product works as a network system where multiple units collaborate to both detect gunshots and triangulate the location of the shooting. The results section clearly shows how quickly our system can detect a gunshot and alert the proper authorities.

A. Current Market

Smart gunshot detection systems are becoming increasingly common in urban cities [3]. Large cities like New York CIty have a policing budget of over $4 Billion. Many cities currently employ a gun detection system called ShotSpotter. This system is a first step in the right direction; however, their system has a high false positive rate and reporting delay of nearly one minute. [4]. Despite these shortcomings, more than 90 cities around the world are using ShotSpotter.

Recently, Durham, North Carolina received a proposal from ShotSpotter to install gunshot detection technology in a three mile radius. Twenty to twenty-five sensors would be installed per square mile. The initial annual cost was estimated to be $235,000 and then $195,000 annually thereafter [5].

If we had submitted a competing proposal to Durham, we would be able to place two-hundred nodes and remain significantly cheaper. We have calculated an approximate initial annual setup of $50,000 and approximately $2,000 annually thereafter to cover cloud computing costs. This leaves plenty of room for profit for our investors.

II. Target Problem

In order to implement successful and useful gunshot detection our system will focus on achieving two main goals.

1) Detecting when a gunshot has occurred. When a gunshot occurs, the system needs to be able to detect it immediately and precisely.

2) Tracing the location of the gunshot. This allows the first-responders to know, roughly, where the gunshot occurred, to potentially save a victim's life.

III. System Overview

Here we will discuss the technical implementation of our product. We propose a network of many nodes centered around a few receiving Multitech Conduits.

A. Individual Module Overview

Each node will have a unique identifier and three main physical components.

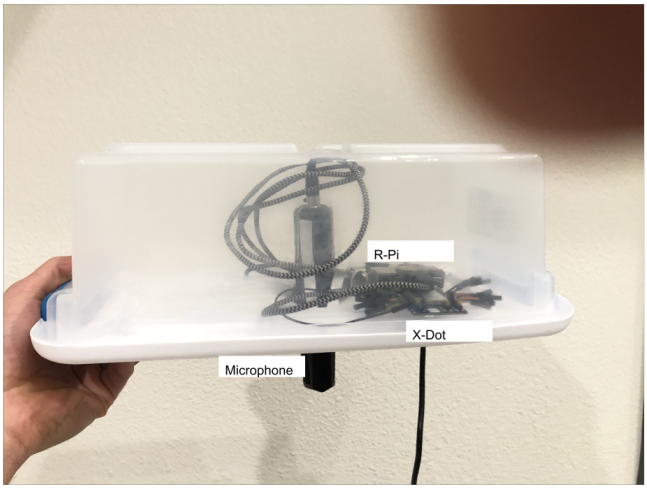

Fig. 2. Physical components assembled attached to lid of a box.

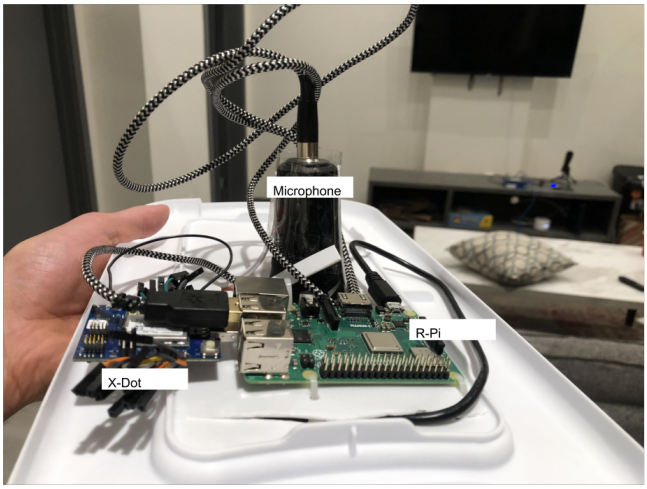

Fig. 3. Physical components assembled attached to lid of a box.

1) Microphone Audio Sensor
2) Raspberry pi
3) Multitech xDot

Our original design included just the xDot and microphone sensor but we quickly found that the memory storage space was not sufficient for performing ML analysis on the edge. If predictions do not occur on the edge than our system would bbe required to constantly stream data to the cloud for near-real time detection. This is simply not realistic in terms of bandwidth usage, especially as our system scales up. For this reason we implement a Raspberry Pi so we can do ML detection on the node.

Our Raspberry Pi program sets the microphone to 16KHz sampling rate with 16 bit resolution and mono output. The program then reads the raw audio data stream into a buffer which stores up to 1 second of data. The buffer's size remains constant and any new data that gets pushed in overrides the oldest data. As the data is read in, the program checks if audio amplitude exceeds a predefined threshold. This threshold was determined by analyzing audio data that could be considered noise and comparing it to gunshot audio data.

Once the threshold is surpassed, the audio data stored in the buffer will sent through the YAMNet machine learning model which outputs the top 5 predicted sounds out of the 512 possible labels. The model will also output the corresponding levels of confidence for each of the top predicted sounds. If gunshot is among the top 5, we trigger a gunshot detected event. We have made use of multi-threading so that we do not miss audio data while we are performing our prediction.

The ML algorithm takes about 0.4s to execute and during that time, data continues to be read. If the threshold is crossed while the data processing is taking place. The ML algorithm is immediately triggered after it is done completing. This can work continuously as long as the window duration is ¿ 0.4s. If it is greater, then the event will be marked as missed and the ML algorithm will not be immediately triggered.

If a gunshot was indeed heard than that information is sent to the xDot over a USB serial port. The xDot is running code that reads from its serial port, transmits the data via the sx1276 LoRa radio, and replies over the serial port when it is ready to send again. This handshake guarantees that the information about the detected gunshot is sent to the conduit.

B. Network Overview

Fig. 4. The mBed xDot, used for its LoRa radio.

A single network consists of several individual nodes around a multitech conduit pre-loaded with Node-Red. The conduit receives two main pieces of information from the xDots over lora radio: the confidence of a detected gunshot and the timestamp at which it was detected. By sending data over lora radio instead of another method like wifi we get two main benefits, longer range and lower costs.

- Higher Range
  Lora signals have a range of 10 km if there is direct line of sight. We will be able to achieve that line of sight because our product will be placed high up out of the way of everyday life. This means that we will not need to have too many conduits to cover a very large area.
- Lower Costs
  Other competing systems utilize wifi to post their data. This leads to a higher recurring costs as a SIM card is needed for each node.

On node-red the data is converted to a json object and then sent to the cloud. The only requirement to connect to the cloud is to create a HTTP request node with the URL being the endpoint from the AWS account

IV. mBed xDot Code

```
    // Set the Data Rate to maximum
    if (dot->setTxDataRate(max_dr) != mDot::MDOT_OK) {
        logError("failed to set TX datarate to DR1");
    }
    pc.printf("TX DR: %i\r", dot->getTxDataRate());

    int ARR_SIZE = dot->getNextTxMaxSize(); // get the max packet size
    ARR_SIZE -= 5; // for the commas & colons in there
    char arr[ARR_SIZE];
    int sig_start = 0;
    std::vector<uint8_t> tx_data;
    while (true) {
        // clear the array and vector
        memset(arr, 0, ARR_SIZE);
        tx_data.clear();

        char c = 'f';
        pc.printf("ready\n");

        for(int i = 0; i < ARR_SIZE && c != '!'; i++){
            c = pc.getc();
            arr[i] = c;
        }

        for(int i = 0; arr[i] != '!' && i < ARR_SIZE; i++) {
            if(arr[i] == '!'){
                break;
            }
            tx_data.push_back(arr[i]);
        }
        send_data(tx_data);
    }
    return 0;
}
```

Fig. 5. The mBed xDot, used for its LoRa radio.

We did not use the xDot for much more than its LoRa radio in this project. In order to have direct access to the radio from the Raspberry Pi, we establish a serial connection between the two. Then on the xDot side, we run a loop wherein we are polling to see if the Raspberry Pi is sending us information to be transmitted. When the xDot receives this information, it will transmit the packet. When it is done, it will alert the Raspberry Pi that it is ready to send again.

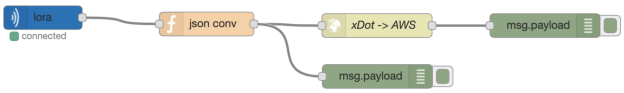

Fig. 6. Overview of the Node-RED flow.

V. AWS Cloud Overview

Our project makes use of a handful of AWS applications: IoT Core, Lambda, and S3. IoT Core is an application that receives messages from IoTs and publishes those messages to an MQTT topic. Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code. S3, or Simple Storage Service, is a service offered by AWS that provides object storage through a web interface.

Data from the conduit is passed over to IoT Core. In the IoT Core all messages sent by the xDot are available in the topic in JSON format. The object contains the device's embedded unique ID (EUI), timestamp, gun-shot confidence and the volume. However, these messages are not saved by default and need to be saved into S3. To save messages into S3, we invoke a rule, that saves a given message to S3. It is possible to save the files such that the files are named [Device EUI].json. By doing this, all messages stored in the S3 Bucket are current JSON objects and this allows us to iteratively search through the bucket. Searching through all the JSON files allows us to find the three nodes that heard the gunshots first to pinpoint the location of the gunshot.

A. Node-Red

When the data is sent over from the xDot/Pi system, before it gets to AWS, it needs to be fitted for AWS. The Multitech Conduit comes pre-installed with the Node-Red application, a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things. Node-RED provides a web browser-based flow editor, which can be used to create JavaScript functions. The flow, which can be seen in Figure 5, receives the data from LORA and then feeds into a function that "JSON-ifies" the string, outputting a JSON object with attributes. This data is then fed to another node that sends the data to AWS IoT Core. This node is an HTTP request node that given the endpoint of the AWS and the certificates needed, it establishes a connection with AWS, allowing messages to be sent to the server. After the data is sent, AWS handles the rest.

B. Multilateration

To pinpoint the location of the gunshot, we need to run a function. Lambda is perfect for this because it can be set to only activate based on a trigger. The trigger is a condition that,

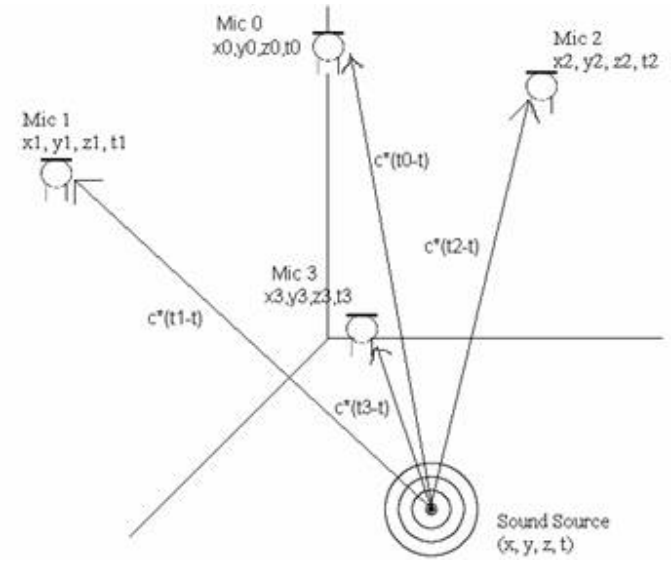

Fig. 7. Multilateration using audio data.

if met, calls the function, giving us our desired output. The trigger we set is that if a gunshot message is heard on IoT Core, activate the lambda function. Our function iteratively finds the three nodes that heard it the soonest and grabs the timestamp and EUI of the nodes. The location of the nodes is saved in a lookup table referenced by the EUI. By using the EUIs, we can find the GPS location of the nodes that heard the gunshot. The GPS location is in standard WGS84 format and is then converted to Cartesian coordinates. After converting the coordinates, we run a multilateration algorithm based on Bancroft's Algorithm [7]. This algorithm makes use of differences in Times of Arrival (TOA) of a signal and locations of base stations (our nodes) to pinpoint the location of the sender of the signal. In our case, the signal is a sound wave and the velocity of a sound wave is 343 m/s. By using this, we can accurately obtain the Cartesian coordinates of where the sound originated and convert it back to GPS coordinates.

C. Data Usage

After Lambda generates the results we need, AWS doesn't give a simple way to do anything with the data. So to make the data useful, we incorporated two systems to make the data more readable. First, we implemented an alert system via AWS Simple Email Service (SES). SES allows us to create a mailing list that can be subscribed to. This is implemented in the Lambda function as python code and it calls the SES service which then emails the coordinates of the gunshots to anyone subscribed. Secondly, we created a mapping system. Given the coordinates of the gunshot and using the gmplot library in python, we were able to create an html file that marks the location of the supposed gunshot location on a google map. These two systems allow the user to more actively visualize and use the data.

VI. Time Synchronization

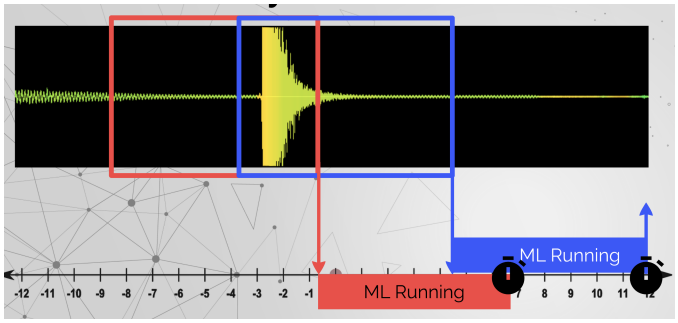

Fig. 8. The gunshot can be anywhere in the window.

Initially, we were going to use the built in time stamps that go along with the LoRaWAN protocol. However, we soon realized that there could be inaccuracies on the order of seconds due to where the gunshot lies in the sliding window. If by chance, the gunshot is at the start of the window on one node and the end of a window on another node, we would detect a multiple second time difference, which is incorrect.

To fix this, we normalize the clock on the Raspberry Pi to the Master clock running on AWS. To do this, we send the Raspberry pi time almost instantaneously, and at least with equal delay at every node, to the conduit. There we calculate the time difference between that node's time and the master time. We store this conversion factor in a lookup table, which allows for easy conversion to real time whenever we receive a Raspberry Pi time stamped message. The Raspberry Pi stamps a window with the time at which the peak sound (assumed to be the gunshot) was detected.

VII. RaspberryPi Code

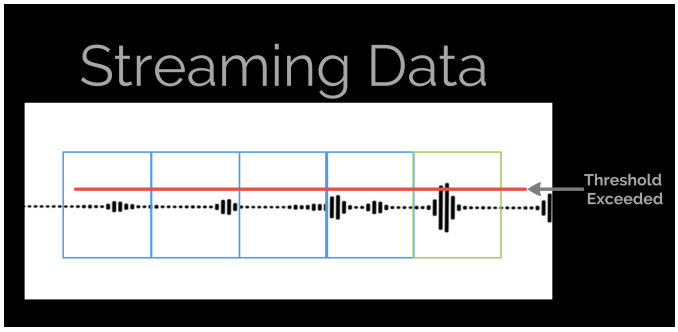

Fig. 9. The audio samples must cross a threshold to trigger the thread to process the audio.

A. Audio Processing

We used a pre-trained model called YAMNet [8]. YAMNet is a pre-trained deep net that predicts an audio event class based on a the AudioSet-Youtube corpus (a massive collection of manually annotated audio samples). The following describes how YAMNet was trained:

- All audio is resampled to 16kHz mono

- A spectrogram is computed using magnitudes of Short-Time Fourier Transform with a window size of 25ms, a window hop of 10ms, and a periodic Hann window
- A Mel Spectrogram is computed by mapping the spectrogram to 64 mel bins covering the range 125-7500Hz
- The stabilized log Mel Spectrogram is computed
- These features are framed with 50% overlap.

The following paragraph, taken from YAMNet's README explains training the model using this extracted data:

> "These patches are then fed into the Mobilenet_v1 model to yield a 3x2 array of activations for 1024 kernels at the top of the convolution. These are averaged to give a 1024-dimension embedding, then put through a single logistic layer to get the 521 per-class output scores corresponding to the 960 ms input waveform segment."

B. Context Filtering

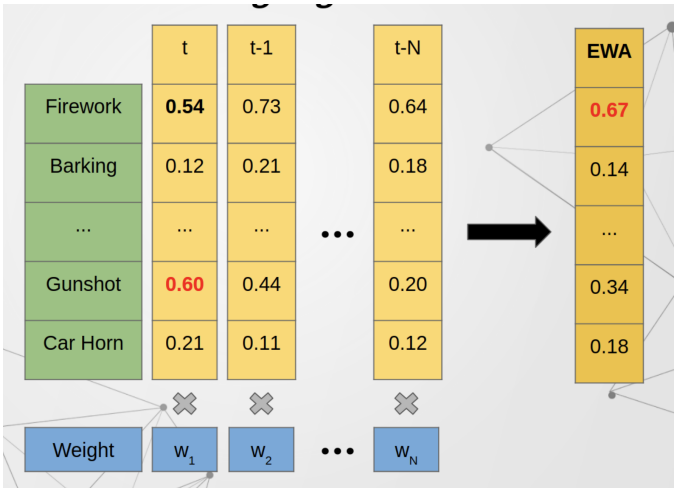

Fig. 10. Filtering outputs from the Machine Learning model with a weighted moving average.

There is an additional filter that we can apply to our gunshot detection algorithm to protect against falsely detected gunshots from fireworks [6]. Since fireworks occur over long periods of time, if fireworks have been detected in the recent past, it should be factored into the decision to report a gunshot. It works by keeping an exponentially weighted average of the N previous prediction confidences. If a gunshot is predicted for an instance in time, it will check against the weighted average for firework predictions, if the gunshot prediction confidence is higher, then report the gunshot, or else ignore it.

In this way, we can minimize the number of false positives that we report during times when there are lots of sounds that might be mistaken for fireworks. During a firework show, for instance, the model would have to then predict a gunshot with very high certainty to override the context filtering algorithm.

VIII. Results

A. Gunshot Detection

a) False Negative: To verify our system can detect real gunshots we visited the Los Angeles Gun Club. While there we ran our system for ten minutes and recorded the results. During those ten minutes a hundred-nineteen gunshots were fired. Our system successfully detected a hundred-thirteen of them and only detected six of them as fireworks.

TABLE I

False Negative Rate	0.0504
True Positive Rate	0.9496

b) False Positive: In order to test our false positive we decided to use Youtube audio. The most common cause of false gunshot detection is fireworks. We played a 10 minute video of nonstop fireworks going off and checked to see if our system detected gunshots instead.

As previously mentioned we decided to implement context filtering for firework audio. Without the context filtering our system detected gunshots four times out of a hundred-sixty-seven. However, if we implement context filtering we detect zero gunshots. Our false positive rates are detected in table II . On firework heavy holidays such as Fourth of July or Memorial Day context filtering can be applied to lower the false positive rate and then turned off afterwards.

TABLE II

False Positve Rate without Context Filtering	0%
False Positve Rate with Context Filtering	2.395%

B. Triangulation

We first tested the triangulation on AWS with fed data. As previously mentioned the triangulation function in Lambda is triggered when an S3 bucket is updated. Once triggered it will iterate through and pull device_id, timestamp, latitude, and longitude for each node. In order to test triangulation we had fixed data in the S3 lookup table for timestamp, latitude, and longitude. The device_ids are generated from Node-red.The values we put are reflected in table III and table IV.

TABLE III
Each device has a unique ID and a detection timestamp

device_id	timestamp
00-80-00-00-04-01-76-64	12:24:49
00-80-00-00-04-01-76-2B	12:24:51
00-80-00-00-04-01-75-EC	12:24:52

The artificial coordinates of Node 1, Node 2, and Node 3 are placed on the University of Southern California campus represent Jefferson & Vermont, Expo & Vermont, and Tommy Trojan respectively.

TABLE IV
LATITUDE AND LONGITUDE OF EACH NODE ARE STORED IN A LOOK UP TABLE

	Latitude	Longitude
Node 1	34.025277	-118.291241
Node 2	34.018477	-118.291364
Node 3	34.020545	-118.285404

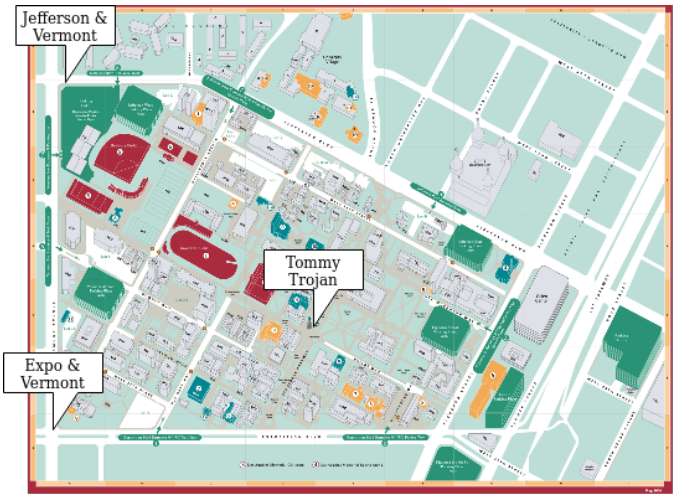

Fig. 11. Visualization of node placement

Our Lambda function successfully pulled in the correct id's and coordinates. From there it used the gunshot detection timestamps to generate GPS coordinates of the likely gunshot location. With this input data our function generates the following:

```
{
  "statusCode": 100,
  "body": "\"where the gunshot occurred\"",
  "lat": "34.022294",
  "long": "-118.289687"
}
```

When the coordinates are plotted we see that it is very close to the center of the three nodes which makes logical sense. The location is not perfect because minute changes in latitude and longitude map into a significant distance difference.

C. Combined System

We combined gunshot detection and triangulation and tested our full system within a small range. Three nodes were placed within a house and we played a gunshot sound. All three nodes detected the gunshot and sent their locations and timestamps to our AWS. From there our Lambda was triggered and we successfully sent an email with the longitude and latitude coordinates of the house.

IX. TECHNOLOGY DEMONSTRATION

In this section, we include images and screenshots of our product at work. This is to demonstrate not only that the product is fully functioning, but to further show the inside workings of our design and implementation.

A. ML at the Edge

We performed audio processing at the edge on a Raspberry Pi. This allowed us minimize how much of the data we would need to transmit over the LoRa radio. The following figure shows the outputs of our audio processing performed on the Raspberry Pi.

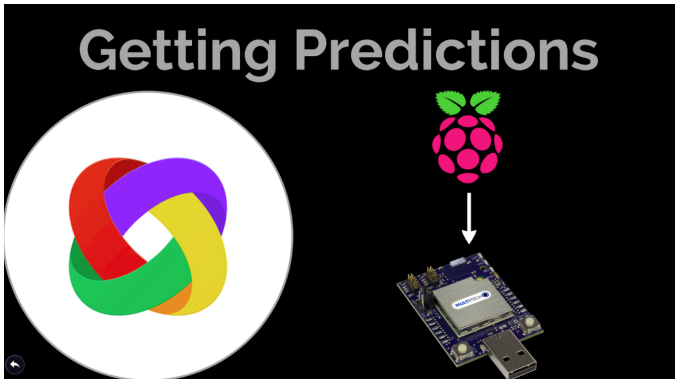

Fig. 12. Visual Diagram of Flow

B. Context Filtering

The images below show the before and after of the implementation of our context filtering algorithm.

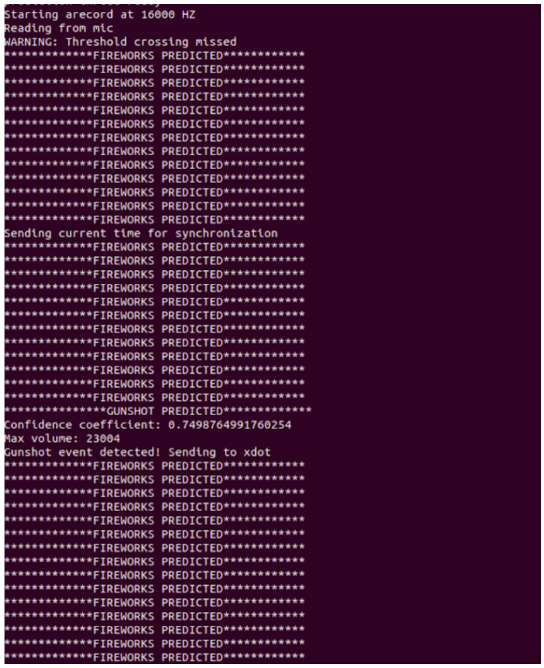

Fig. 13. Before our context filtering algorithm, we can see gunshots are being predicted with the fireworks.

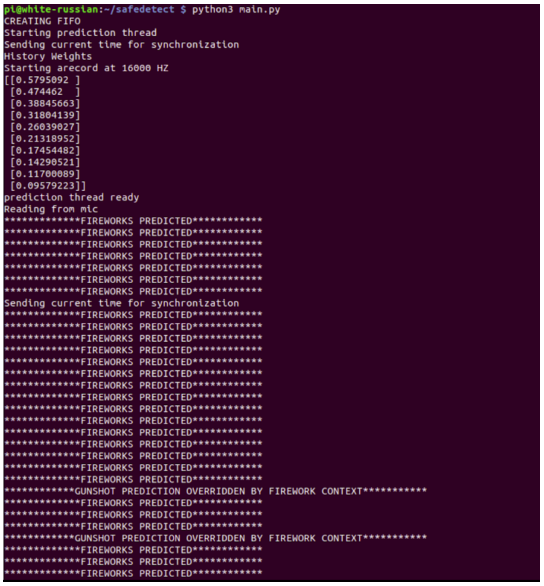

Fig. 14. After our context filtering algorithm, we can see gunshots are being predicted with the fireworks, however, this time they are being overridden because of the recent history of fireworks.

C. Node-RED

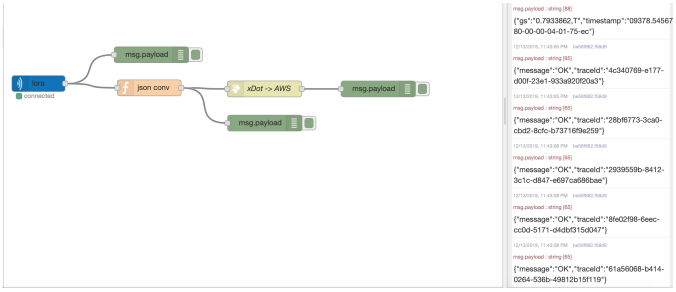

Fig. 15. Here we can see Node-Red retrieves messages from the xDot and then converts it into JSON format. We can also see a confirmation message from the AWS console.

D. AWS

X. Challenges Lessons Learned

The single biggest challenge in developing this product was figuring out what we were going to build and how we were going to build it. We spent weeks prototyping with Analog MEMS microphones. We wrote xDot code to overwrite the LoRa bandwidth sharing policies so that we could stream audio. Doing ML at the edge was the obvious solution to all of these problems. We did not consider all the roadblocks we might face in developing on the xDot. In future works, we will definitely flesh out our ideals more fully from a systems perspective before getting our hands dirty with implementation. It is always worthwhile to verify that you can fit a program on a chip, with plenty of room to spare. In wireless networking, it is crucial to identify the use cases of your radio, and try not to bend this radio to your needs. Instead, get different components.

Fig. 16. After IoT core receives messages, it invokes rules to send the data to Lambda to call the Multilateration Function.

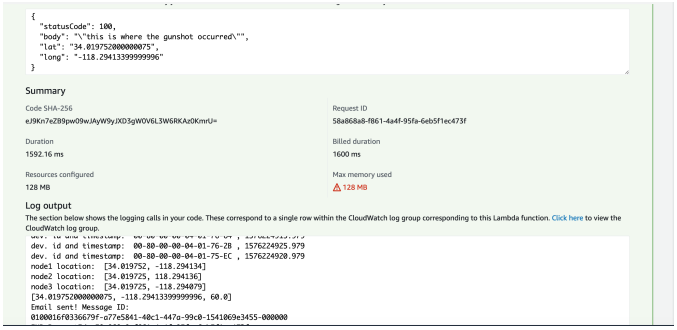

Fig. 17. After Node-Red sends the data, the data can be seen on IoT Core on the subscribed topic.

XI. Conclusion

Gun related deaths are issues in this country that can be mended. Because most gunshots are reported by bystanders, when a bystander does not report on it, often, the victim is left helpless. Our product, through the above mentioned details, combats this issue to alert first responders immediately after a gunshot to help save a victim's life. Our product is currently in the prototype phase. We were able to implement a working prototype that constantly streams audio, performs predictions on the audio, streams to AWS, and sends alerts over email with the likely location of where the gunshot occurred. Our next steps would involve future testing.

XII. Future Work

Ideally our system needs to be tested on a much larger scale where we would be able to place nodes on streetlamps along the neighborhood blocks. In addition we should attempt gunshot detection outdoors with real firearms. Our testing with live gunshots was done within a gun range where the shots themselves occurred in relatively close proximity to our system. In actual implementation we would need more fine-tuning for the audio detection sensitivity at a longer range.

Acknowledgments

We would like to acknowledge our Internet and Cloud Computation Professor, Dr. Young Cho and our Internet and

Cloud Computing Teaching Assistant Arthur Win. Additionally, we are grateful for the YAMNet audio processing machine learning model and the accompanying dataset used to train it. Without this, our predictions could not have been so accurate.

REFERENCES

[1] Centers for Disease Control and Prevention, The Steady Rise Of U.S. Gun Deaths. statista.

[2] I. Freire and J. Apolinario (2010). Gunshot detection in noisy environments.

[3] L. G. Mazerolle, "Using gunshot detection technology in high-crime areas," in Using gunshot detection technology in high-crime areas, 1998.

[4] B. Fraga, "After Too Many Shots Missed, Fall River, Mass., Ends Deal with ShotSpotter," Government Technology State & Local Articles - e.Republic. [Online]. Available: https://www.govtech.com/public-safety/After-Too-Many-Shots-Missed-Fall-River-Mass-Ends-Deal-with-ShotSpotter.html. [Accessed: 13-Dec-2019].

[5] ShotSpotter, "PRICE PROPOSAL FOR SUBSCRIPTION-BASED SHOTSPOTTER®GUNFIRE LOCATION, ALERT AND ANALYSIS SERVICE FOR THE CITY OF DURHAM," cityordinances.durhamnc.gov. [Online]. Available: https://cityordinances.durhamnc.gov/OnBaseAgendaOnline/Documents/ViewDocument/WS-Published Attachment - 13024 - PROPOSAL - 2 SPOTSHOTTER FOR DURHAM - 3_18.pdf?meetingId=296&documentType=Agenda&itemId=10442&publishId=47466&isSection=false.

[6] G. Evangelopoulos, A. Zlatintsi, G. Skoumas, K. Rapantzikos, A. Potamianos, P. Maragos, Y. Avrithis, "Video event detection and summarization using audio visual and text saliency", Proc. IEEE Int. Conf. Acoust. Speech Signal Process., pp. 3553-3556, 2009-Apr.-19–24.

[7] M. Geyer and A. Daskalakis, "Solving passive multilateration equations using Bancrofts algorithm," 17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No.98CH36267).

[8] D. Ellis and M. Plakal, "YAMNet", GitHub repository, Available: https://github.com/tensorflow/models/tree/master/research/audioset/yamnet

ABOUT THE AUTHORS

A. *Charlie Andre*

Fig. 18. Charlie Andre, Computer Engineering Masters student at USC.

Charlie is a first year Master's student at USC's Viterbi School of Engineering. His interests include wireless networking, localization, embedded systems, and robotics. After graduating with a Master's from USC in May 2020, Charlie hopes to work as an embedded software engineer in the San Francisco Bay Area. In undergrad, when he used to have free time, Charlie would enjoy going skiing and camping on weekends.

B. Yutong Gu

Fig. 19. Yutong, Computer Engineering Masters student at USC.

Yutong Gu is an Electrical Engineering Masters student at the University of Southern California. He expects to graduate in 2020 with a focus on embedded software development. His areas of interest include Robotics and IoT. He also received his bachelor's at the same university in 2019.

C. Sharon Guo

Fig. 20. Sharon Guo, Computer Engineering Masters student at USC.

Sharon Guo is a current Computer Engineering Masters student at the University of Southern California. She received her bachelors in Electrical Engineering at USC in 2019 and will be completing her masters in May 2020. Her coursework specializes in embedded computing and computer architecture. She is from Orange County and will be remaining in Southern California as part of the aerospace industry.

D. Sufyan Shaikh

Fig. 21. Sufyan Shaikh, Electrical Engineering Masters student at USC.

Sufyan Shaikh is a current Electrical Engineering Masters Student at the University of Southern California. He received his Bachelors in Electrical Engineering at USC in 2019 and is on his way to completing his masters by the end of 2020. His coursework emphasizes in VLSI design and embedded systems. Throughout his time at USC, Sufyan has been an active member of the engineering community, participating in clubs such as Hyperloop, Race On, and many more. Born and raised in the city of Los Angeles, Sufyan plans to stay in the area after he graduates.