Super sensors Ad-hoc network

Jay Lohokare, Revati Damle

Department of Computer Science,

The State University Of New York, Stony Brook

Motivation:

The modern world appliances are connected and intelligent. However, there are many drawbacks in the existing smart appliance systems. They have redundant components, they are costly, they need dedicated internet connections. There is no notion of edge intelligence and offline analytics possible due to the centralized server-client type of architecture.

The aim of this project is to develop a platform to resolve these issues – a dynamic network of 'super-sensors'. These super sensors should be able to detect any events happening near them, by leveraging machine learning (Convoluted Neural Networks). The network of these super-sensors should be modular, scalable and fault tolerant (Ad-hoc networks).

Such 'super-sensor ad-hoc networks' would detect any event happening in the 'smart-environment' they create, and the range of these networks would theoretically be infinitely scalable; thereby resulting to a fault tolerant, less costly, infinitely scalable ubiquitous general purpose sensor network.

INDEX

Abstract:

‘Smart appliances’ and Ubiquitous networks are on a rise in gaining popularity of the notion of connected smart devices. Such ‘smart appliances’ are able to detect events related to their usage, performance, and outputs; which is possible due to the inbuilt sensors that are shipped with almost every of such devices. Ubiquitous sensor networks, on the other hand, have helped such ‘smart appliances’ talk to each other, thereby performing actions together as a combined entity. This project presents an evolutionary step ahead in smart ubiquitous networks by presenting a novel approach towards sensing based on ‘super sensors’. Instead of embedding sensors into every single appliance, we could build a ‘smart environment’ that can detect any event happening related to the appliances in it. The approach presented involves the use of a single sensing unit that captures multiple sensor streams like Light, Heat, Temperature, Vibrations, Electromagnetic noise, etc and then uses machine learning to detect all possible events that could happen. This ‘super sensor’ will result in a ‘smart environment’, the range of which can be expanded on demand by involving Ad-hoc networks. The project developed uses TensorFlow to build machine learning model over data collected from Android/Android-things. The super sensors developed communicate using a pseudo-AODV protocol based on BLE. Such Ubiquitous smart sensor networks can potentially disrupt the notion of Internet of Things and the connected world by minimizing costs, enabling machine to machine communication, mobile sensing, low energy usage and a dynamic on-demand network topology.

Introduction:

The ever-increasing popularity and use-cases of connected smart appliances have resulted in a boom in the number of smart appliances available in the market today. The concept behind smart sensors is really useful – Appliances collect data related to their performance, outputs, user interactions, etc and then use this data to perform actions based on certain rules pre-fed into these devices. Real-time notification is the most common set of actions that these smart devices are capable of – Sending real-time data feeds to smartphone applications/websites so that users will get notified when a particular event occurs. Most of the actions other than notification are complex as they usually involve interactions between 2 or more devices. For example, a smart-thermostat (Essentially a thermostat with internet connection) can send events data not just to users but also to other devices like AC/Heater. A thermostat will detect the temperature of the room, and then decide if the temperature needs to be cooler/hotter. It will then send a message to a 'Smart AC'/'Smart Fan' to turn off/on. These interactions could be more complex, but for sake of simplicity, for example, we will use this example throughout this report.

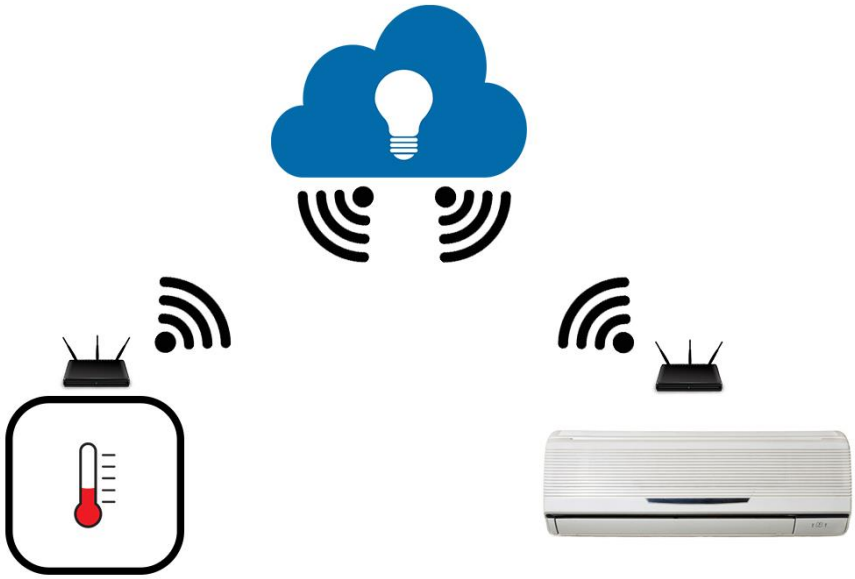

Figure 1: Interactions between 2 smart appliances – AC and a thermostat.

This concept is highly beneficial! Devices can perform actions on their own once we set some rules for them (For example – The thermostat will have a rule to turn AC on only when the temperature is above 70 degrees). But there is a huge scope for improvement in the existing architecture involved.

Let us list the shortcomings of the architecture commonly used in smart appliances –

1. Replication of sensors:

Let's consider a smart appliance "A". "A" can be any smart appliance – Smart oven, smart refrigerator, smart AC, smart thermostat. Keep 2 or more such "A's" in the same room would mean that we have 2 sets of same sensors sitting in a room. This leads to redundancy, there needs to be a way to avoid such replication which will help us reduce costs.

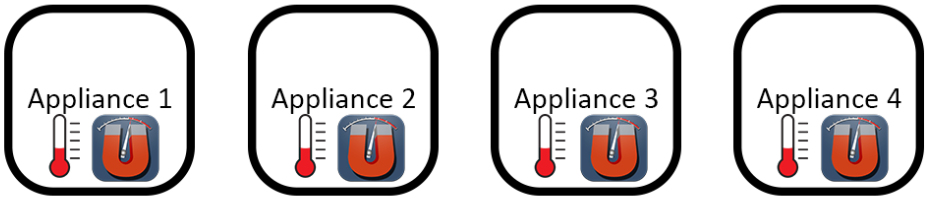

Figure 2: Appliances today – Replication of sensors in every appliance

2. Internet communication:

Any data coming from a smart appliance follows a data pipeline – Data is sent by the smart appliance to cloud, where it is redirected to appropriate application and user. Thus, every single smart appliance is now streaming data to a cloud. If we have 100 smart appliances, there will be 100 concurrent data connections with the cloud. There will be 100 of devices with hardware required for internet connection. This is another aspect of redundancy that should be avoided.

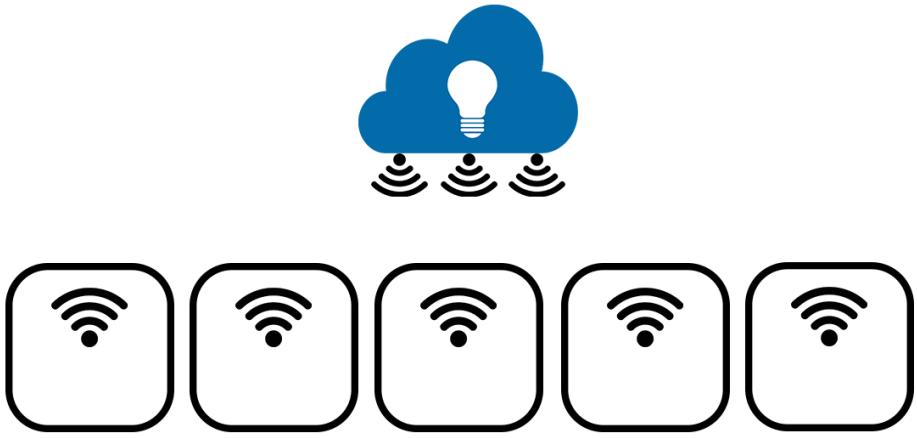

Figure 3: Multiple internet connections – every appliance talks only with the cloud

3. Cloud-centric communication:

The communication between 2 smart devices sitting next to each other has to pass through a cloud. There is no scope for what's called 'Local intelligence' or 'Edge analytics'. This results to a tremendous usage of data, energy and is slow.

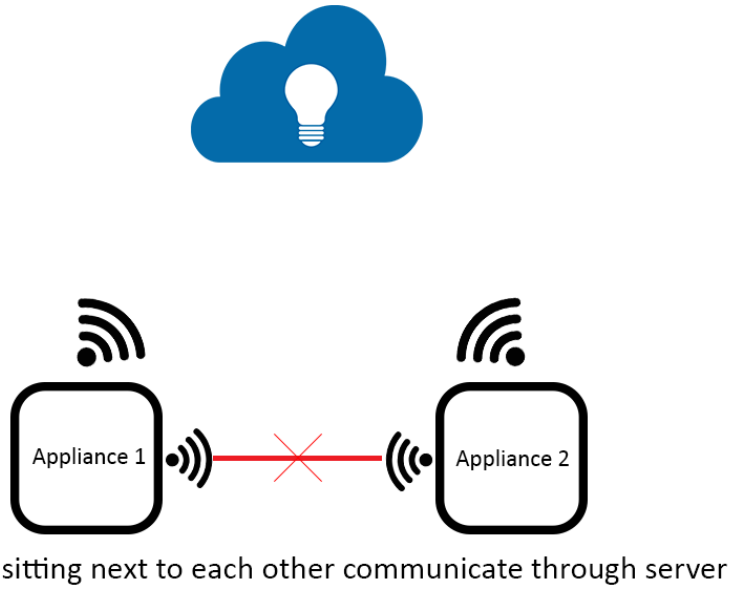

Figure 4: No/Minimal 'M2M' communication

As described before, the current state of art 'smart appliances' have multiple short-comings. These shortcomings are mainly because every appliance manufacturer tries to inculcate intelligence into devices by adding more sensors and processing power into individual devices. This approach is good, but not in long run. There should be an alternative to avoid such a waste of computational and hardware resources. This problem of redundant internet connections and sensors is what motivated this project.

Concept of general purpose sensing:

Today, the term 'general purpose sensing' is existing everywhere in form of CCTV cameras. These cameras capture images/video which records all visual events. A learning model can easily be trained over these images/videos to detect complex events happening in the environment. However, there are a few shortcomings in such a general-purpose sensing camera – Camera can capture only visual data, and cannot detect events that don't result to change in waves in the light spectrum. For example, temperature of room or temperature of a heater can't be detected by cameras. Also, cameras invade privacy, which makes it's a controversial topic to use cameras for general purpose sensing.

Let's get back to the main problem – Multiple sets of sensors in all devices. To eliminate such multiple sets of sensors, we somehow need to build a device that has one such set of sensors, and it can detect events related to all other devices.

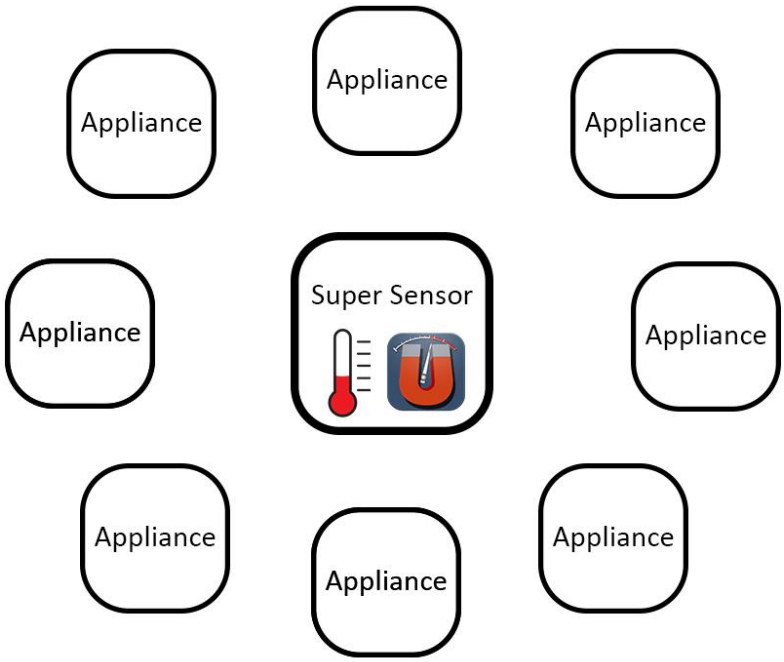

Figure 5: One sensing unit for multiple appliances – The new approach

This approach seems easy, and that's exactly what we have implemented. But the issue that arises is that how can we detect events related to all any device in the 'smart environment' without interfacing with the device.

Super Sensors:

To avoid embedding sensors into every single appliance, and to make sure that events related to every device are correctly detected, we need a 'super sensor'. This super sensor will somehow be able to detect any event that happens in the 'smart environment' it generates. Achieving this is possible by using Machine Learning. The 'super sensor' is a hardware module with multiple sensors and a BLE (Bluetooth low energy) adapter.

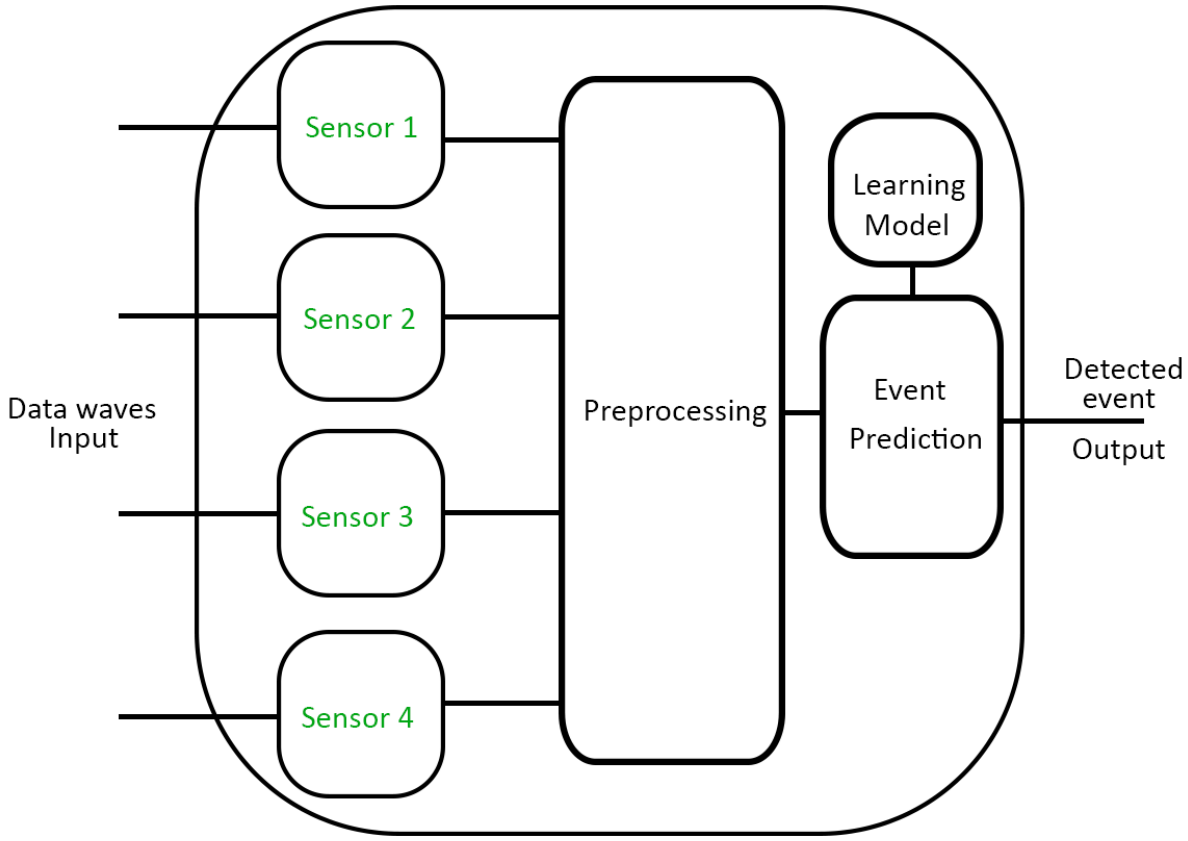

Figure 6: Internal architecture of super sensor

The hardware we propose will have a set of sensors to detect most common data points that can predict events occurring in the smart environment. The choice of these sensors will depend on the appliances present in the smart environment. We will explain the reason for the choice of sensors in later sections. The fundamental idea here is that the hardware module will be able to sense various forms of waves from the environment. Any events occurring in the environment would always generate some forms of data waves. For example, a microwave when switched on generates magnetic noise along with some peculiar sound. Every event that we may want to detect in an environment will always have some or the other waves specific to the event. For example, opening a microwave oven's door will generate just some sound but no magnetic noise,

while a microwave when heating creates some magnetic noise along with some other peculiar sound. Detecting these data patterns can basically detect events happening in a smart environment. This applies to every event related to every appliance in the 'smart environment'.

Now, an argument that can be made here – Just one sensor with minimum data entropy can detect all events. For example, mic (sound) sensor can alone be used for event detection in a smart environment. This is true for most of the events we discussed above – tap turned on, microwave oven door opened, microwave oven powered on, etc. However, there are 2 shortcomings which makes it hard to use mic alone for creating a smart sensor. Just like we discussed how a camera can't detect all events happening in an environment (Because not all events have a visual data generated), not all events may have a sound profile (For example – temperature change can't be detected by a mic). The second shortcoming in using just one sensor – Let us continue with an example of a mic being the only sensor used in the smart sensing module. This mic will blindly trust the sound data points. If we record and play the sound of microwave oven, the mic will falsely detect it as a microwave turned on. Thus, using lesser number of sensors is not a good idea, as the system generated will have multiple faults and loop-holes. The following diagram shows how multiple sensors coming together can detect an event happening in smart environment.

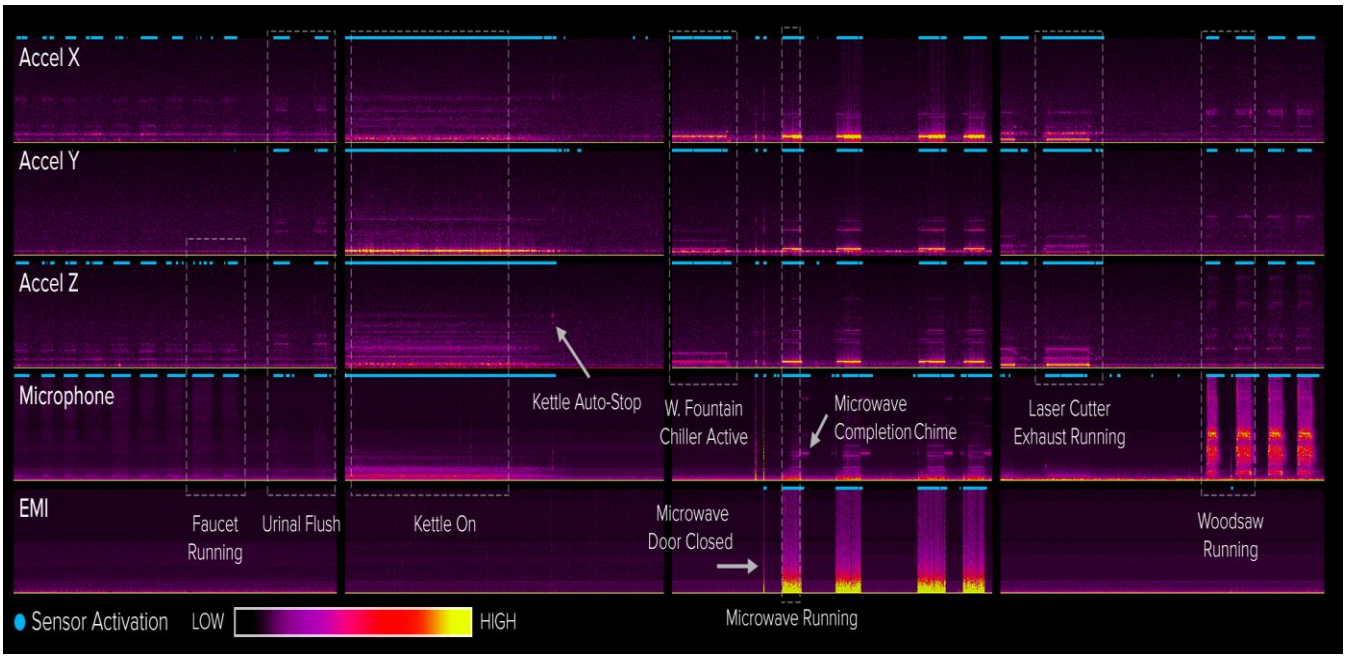

Figure 7: Sensor activations for various events (Source: https://dl.acm.org/citation.cfm?id=3025773)

Now that we have established how data captured from multiple sensors are potentially able to detect events in an environment. In next section, we will describe how to build the learning model.

Machine learning – Differentiator between a sensor and 'super-sensor':

The data points captured are of multiple formats, multiple distributions. For example, temperature sensor captures a numerical value depicting the temperature in Celsius/Fahrenheit, whereas the mic captures a multi-dimensional data – let's say it captures sound into a raw (.wav) file. Imagine constructing a simple rule-based decision tree using just 2 sensors – Temperature and mic. Temperature data being a number could easily constitute decision tree node. To elaborate more, temperature will detect if it's hot or not. Making this decision is simple – If temperature is above some value, it will be classified as hot. However, this same model construction technique can't apply in-case of data from a mic. To classify between a sound of microwave powered on and microwave door opened, we can't build a simple decision tree. The classification will be based on a time-series data. The sound data reading at a given moment alone can't predict the event; we will need sound of at-least 1 second long to make any sensible learning model. All these complications are when we consider just 2 sensors. With 8-9 sensors acting together, the learning model will get too complex. An ideal way to train such a model will be by using Timed series neural networks like LSTMs.

Another factor to understand how data contributes to the learning model is the way the data changes as per events. For example, the data captured from temperature sensors itself can help the model to detect events (Like the environment being hot/cold). However, the data readings from a magnetometer don't depict anything by itself. A magnetometer is used to detect events like microwave powered on – It is not the actual readings of the magnetometer, but the noise in the sensor readings that helps in the detection. The magnetic field generated by powering on the microwave causes fluctuations in magnetometer readings. Thus, variation in magnetometer readings is the data that should contribute to the learning model. Thus, for some sensors, the actual sensor readings contribute to learning, while for others, derivatives of various orders can contribute.

The experiment we conducted to prove this model involved use of 2 sensors – Mic and magnetometer. We captured sample data by building an android application that dumped the magnetometer data points into a file and generated a WAV file. We built learning model for 2 events – Tap water turned on, and microwave turned on. The data capture application had 2 buttons, one to start the recording and another to stop. The model we used to learn was the following:

We considered a sample of 1 second as our input. The mic recorded WAV files 1 second long, separated. While capturing the training data, we kept data points 5 milliseconds apart, to construct a good training set. The WAV file was direct input to the learning model. For the magnetometer, we captured the sensor readings separated by 5 milliseconds.

We used the Tensorflow framework to train this model. As discussed earlier, we could create a single model with multiple data points as input parameters (This will be the ideal model). However, creating an LSTM over such data will need high precision sensors, and sensor readings

separated by not more than a millisecond. As we had access to just the Magnetometer and Mic in Android phone, this was not possible. Hence, we devised another equally better model for predicting events. We ensembled 2 learning models:

1. The first model predicted the event based on just sound

2. The second model predicted if microwave is turned on/off using the variance in magnetometer readings.

As the magnetometer data predicted just into 2 classes – Microwave turned off, Microwave turned on; this model was a simple single node decision tree (rule-based inference).

These models ensembled together gave us a highly accurate prediction of events!

Steps involved in building a super sensor:

1. Creating prototype for hardware with sensors to collect data for training the learning model:

We started trying out a RaspberryPi based sensor data capture code for capturing data from mic. The python code generates WAV files of input sound. Then, we decided to migrate to Android smartphones for capturing other data as lacked sensors for RaspberryPi (Magnetometer). We created an application for capturing sound (WAV files) and magnetometer data which generated a file with magnetometer readings and WAV files. The main reason for choosing Android as our primary platform is that the growing popularity of android-Things and the portability of code between Android and Android things makes it easy to deploy applications on smartphones as well as on hardware with minimal changes. Thus, the super sensor could be a smartphone application or a hardware sensor, and we can port the same codes to both the platforms!

2. Pre-processing the data:

The data captured from magnetometer and mic was in formats that couldn't be fed directly to the learning model. The sampling rate and quality of sound captured had to be changed, and the magnetometer data points had to be converted to variance. Now, usually, if we design an LSTM, it will itself recognize that 1st derivative of the data is contributing to the classification. But, as we couldn't use LSTM (Due to mentioned reasons) and were using ensembled models, we had to manually convert the data to variance. For reasons explained later, we performed the data pre-processing step on Android.

3. Training the model:

The data after pre-processing helped us build a classification model. We used TensorFlow as framework for Machine Learning. The training involved passing the WAV files and magnetometer data file to the classifier models. The 1st model of predicting sound needed negative samples (Sounds other than microwave powered on and tap turned on). We used 'Silence' as one such prediction class and 'Background Noise' (From Google sound dataset) as another. This helped the model have negative samples as well as positive ones. The model we used was a Convolutional Neural network, where the sound data is basically converted into an image, and then we run image classification problem over it.

The core of the model consists of an LSTM cell that processes one word at a time and computes probabilities of the possible values for the next word in the sentence. The memory state of the network is initialized with a vector of zeros and gets updated after reading each word. For computational reasons, we will process data in mini-batches. In this example, Every recording in a batch should correspond to a time t. TensorFlow will automatically sum the gradients of each batch and give the prediction.

The magnetometer variance readings helped us detect if microwave is on or not. As the variance data is a simple Boolean classifier, we created a single node decision tree (Which turns out to be a rule-based algorithm) for classification.

The reason to choose TensorFlow as the framework of choice is that it supports portability of models to multiple platforms. The model once trained on a PC can be dumped to a file and transported to platform of choice. TensorFlow being a Google-powered product, interfacing Android/Android-Things with TensorFlow is highly supported. Also, Tensorflow has the most number of models provided as inbuilt features, which makes it easy to build ensembled models. This made TensorFlow the obvious choice for training the model.

The training step needs high computational resources; hence we perform this step on PCs. The training data once captured on Android, is then copied to a PC with TensorFlow. The training codes are all in Python (Python being the most supported and documented language for Tensorflow). Training the model on CPU of even the latest processors takes around 4 minutes (Considering 3000 wav files per class). This could be made faster by using the GPU accelerated version of TensorFlow (We didn't try this in our experiment, but it's an interesting feature of TensorFlow).

4. Porting the model to a super sensor:

The TensorFlow training on PC creates a learning model. TensorFlow allows model portability by letting users create a model file. This file once created can be ported to platform of choice (Android in case of our experiment). TensorFlow has provided interfaces on Android/Android-things (And many other platforms) that take data points and model file as input and give the output as result. These interfaces make it very easy and less cumbersome to test the model learned. The actual classification after building the model will need the data to be pre-processed and then provided to the model. That's the reason we had performed data pre-processing earlier (after capturing training data) on Android. Implementing the interface classes in the application lets the application (Or hardware) gracefully predict the events happening in the environment.

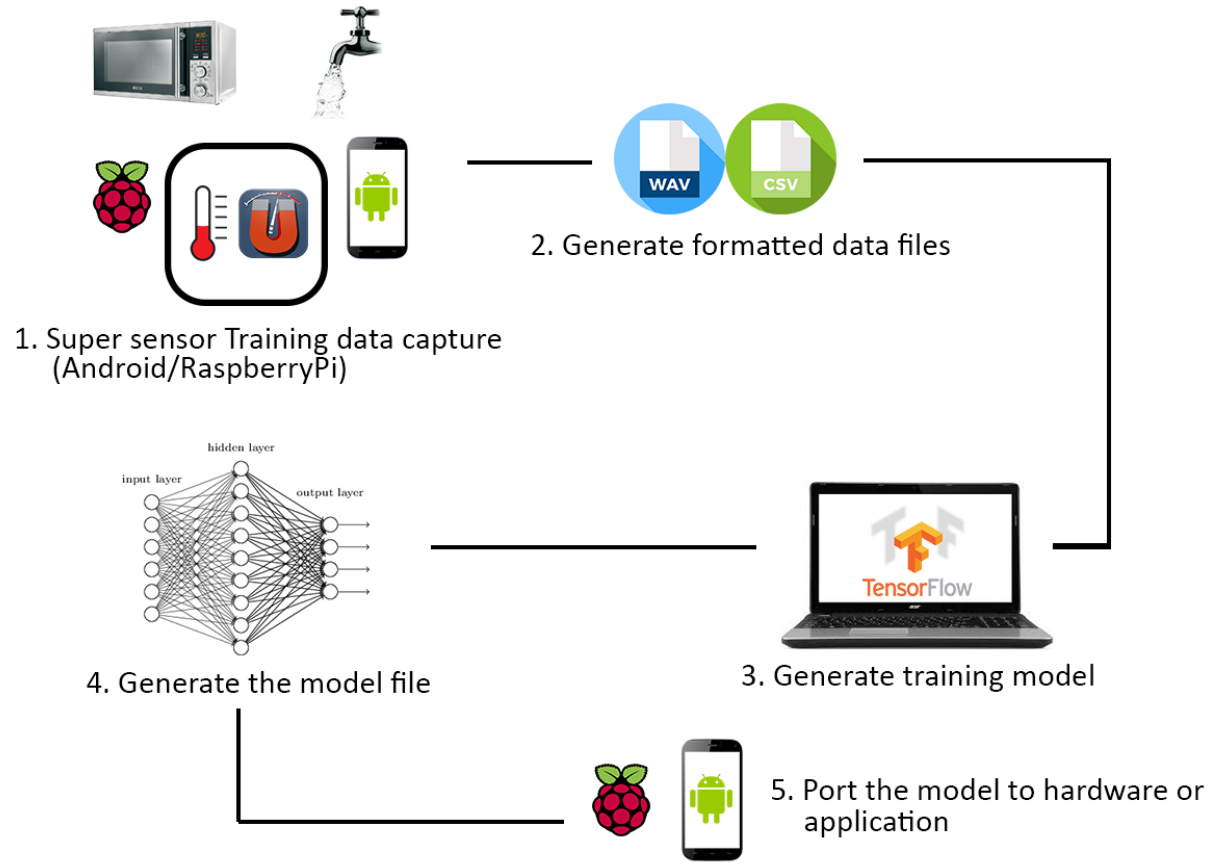

Figure 8: Steps in construction of a 'super-sensor'

'Super-sensor' limitations and constraints:

A super sensor constructed in the way previously explained can detect any events occurring in an environment! The range of this 'environment' is limited by various factors like attenuation of signals with distance, background noise interfering with waves, blocking of signals due to other objects between source of signal and the super sensor, etc. one super-sensor can typically handle events detection. A super-sensor usually fails to detect events if there is a wall between the location of event and the super-sensor. Thus, the super-sensor typically has range of a small room. Considering a stand-alone super-sensor, the super-sensor will transmit the data (detected event) to cloud through traditional internet connectivity mediums. Thus, though we eliminated multiple sensors from a room, we will still need multiple internet connections for multiple rooms. Also, these super-sensors will be connecting to internet via some wireless or wireless medium – This prohibits the super-sensors from being mobile.

Adhoc networks - Overcoming the limitations of super-sensor:

The main shortcoming of the super-sensors is limited range. The obvious way to overcome this is by building a mesh of such super-sensors. The typical sensor networks have fixed network topology, which makes the nodes non-mobile; and adding new nodes requires new routers in the network. Thus, traditional network protocols result in dependence on routers (network topology).

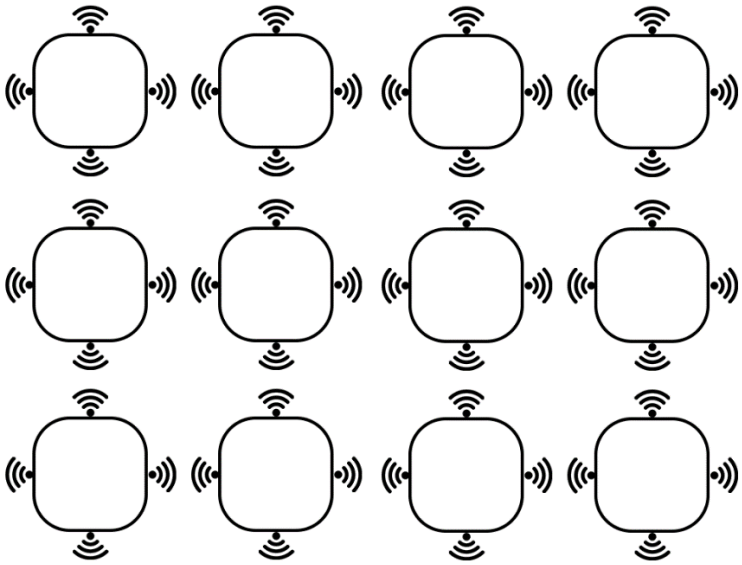

Figure 9: Ad-hoc networks – Neighbour discovery

To overcome these limitations, our sensor network needs to be able to update the network topology on demand. The concept of Ad-hoc protocols is exactly what we need – The nodes in the sensor network can detect new neighbours when network topology changes. This is achieved by a node broadcasting its presence from time to time (The time of broadcast is determined by what exact protocol we use). This makes the network dynamic, thereby eliminating need for physical routers. The routing tables are updated as the network topology changes.

AODV Protocol – Introduction:

The AODV (Ad-hoc On-demand Distance Vector routing) protocol is a protocol developed in Nokia Research centre, in collaboration between University of California Santa Barbara and University of Cincinnati (https://www.ietf.org/rfc/rfc3561.txt).

The fundamental idea in AODV is that every node broadcasts the messages it receives until the message reaches the destination. Every node in AODV protocol implementation has a unique ID. The routing happens by mapping a source and destination ID with every message. The protocol also has a notion of 'hop count' maintained in every broadcasted message. This hop count is increased by one every time non-source/non-destination node broadcasts a message. This eliminates infinite loops of message broadcasting, as a message is broadcasted only if it's hop count is less than the max hop count allowed.

For example:

Suppose there are 3 nodes in the network – 'A', 'B', 'C'.

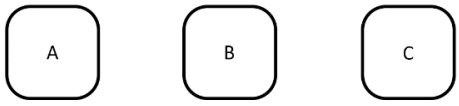

Figure 10: Sample topology – 3 nodes in the Adhoc network

AODV protocol is distance vector-based routing – Let us consider a topology where A and C are out of each other's broadcast range, while A, B and B, C are in broadcast range. Thus, the routing tables for the nodes for given topology look as shown in the diagram. Now, when A wants to send message to C, it will generate a message with sourceID = A and destinationID = C. Also, the hop_count is set to 0. A then broadcasts this message, and its received by B. B broadcasts this message to its neighbors (A, C) by increasing the hop count by 1. C receives the message and recognizes that the message is meant for it (by reading the destinationID).

An infinite loop occurs when message loops in the same path continuously – ABABABAB…

This is avoided in AODV as message isn't broadcasted once max hop count is reached.

Pseudo-AODV implementation over BLE:

Given the experimental setup, we implemented a pseudo-AODV protocol to allow on-demand ad-hoc routing. The super-sensors can use BLE (Bluetooth Low energy) to base the AODV protocol on. A BLE beacon has an 'advertisement' message that it can broadcast to all nearby Bluetooth devices. The key here is that every sensor node is a BLE beacon as well as a BLE receiver.

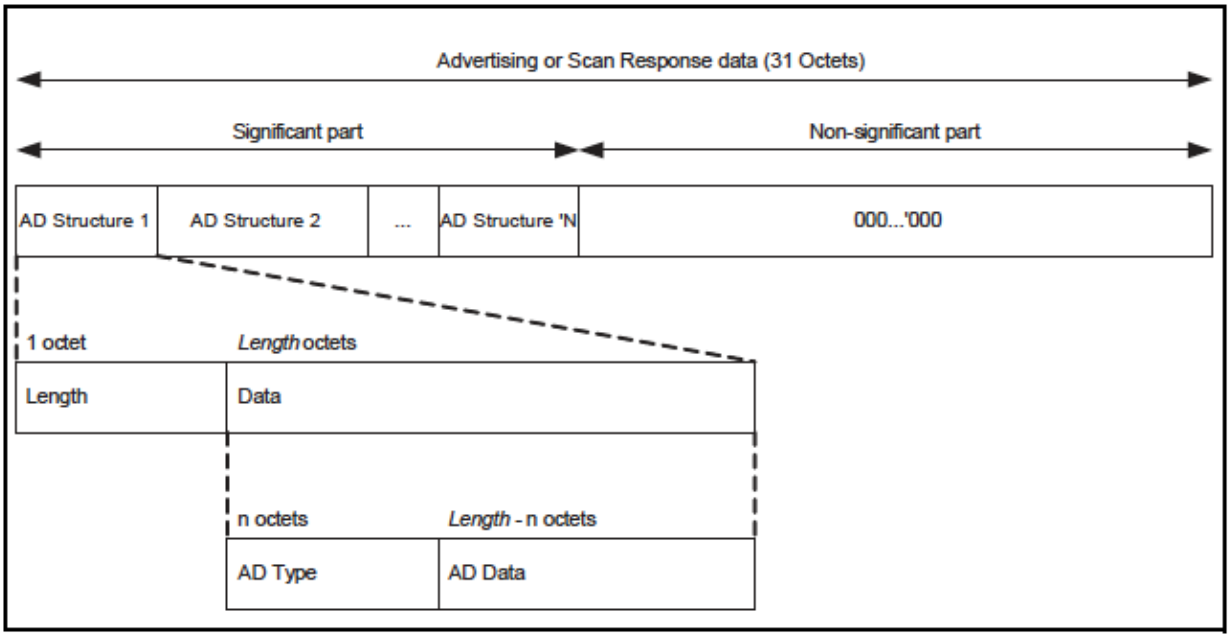

Figure 11: Packet format of BLE Beacon advertisement message

The above diagram shows how a typical BLE Beacon advertisement packet looks like (Actual packet might differ from beacon to beacon – There are multiple standards present). The Ad-data part of the packet is where we will be encoding our message. We have 32 bytes to encode the message, as the Ad-data part of the packet typically allows 32 bytes size. We can encode the sourceID, destinationID, hopCount of all, in these bytes.

The messages can be broadcasted using BLE beacon's advertisement message – Whenever a new message is generated or received from a neighbor, the super-sensor node will increase the hop_count in the message by 1, and create a new BLE beacon advertisement with the new data. All of its neighbors being BLE receivers, will get the data and repeat the same action till the message reaches the destination.

In the experimental setup we constructed, the Android application was supposed to implement this AODV protocol. We utilized Android's BLE class and BLE Beacon class to achieve this. We implemented a 'Pseudo AODV' – essentially AODV protocol with message formatted as a JSON.

Instead of encoding the message into the packet by converting the data to bits, we constructed a JSON and then converted the JSON to bits and fed it into the Ad-data part of the BLE advertisement packet. A sample of the JSON we used:

{

'id' : 1234

'sourceID' : 'A',

'destinationID' : 'C',

'hop_count' : '3'

'event': '12'

}

The message contains event data as a number. These numbers basically depict the type of event recorded by the super-sensor.

How will the super-sensor network work? Let's consider a building with many rooms for which we want to build a super-sensor network. There will be multiple super-sensor nodes separated by distance less than range of BLE, ensuring at-least one super-sensor node in every room. This will make sure that no node in the network gets isolated from the network, and that we have enough coverage to detect events from anywhere in the building.

Without an Ad-hoc network, we would have needed every super-sensor node to have internet connections. With the Pseudo-AODV implementation, we will need just one node with internet connection (Let's call it the 'controller' node). All other nodes will send the event data they record over the Pseudo-AODV with this internet connected node's ID as the destinationID. Suppose the controller node is at ground floor of the building, and an event is recorded in a corner room on top floor. The data from that room's super-sensor will be broadcasted to all its neighbors. This data will keep getting broadcasted by every node, and will eventually reach controller node. The message received by the controller node will contain the senderID of the sender node along with the event data in form of a number. The controller node will typically be a stronger processor, as it will have to handle data from many nodes and in addition to it, send it to servers via internet. This node will maintain a map of all senderIDs to the rooms where those senders belong to, and a map of all event numbers so as to know which event was recorded.

The ID field in the JSON will help avoid data echoes being treated as multiple events – Same data might reach the controller node multiple times through each of its neighbors. The ID field will help the controller node block out messages that it has already received.

The controller node can send the data coming from all the super-sensor nodes in the sensor network to cloud services to perform actions, notify users, etc.

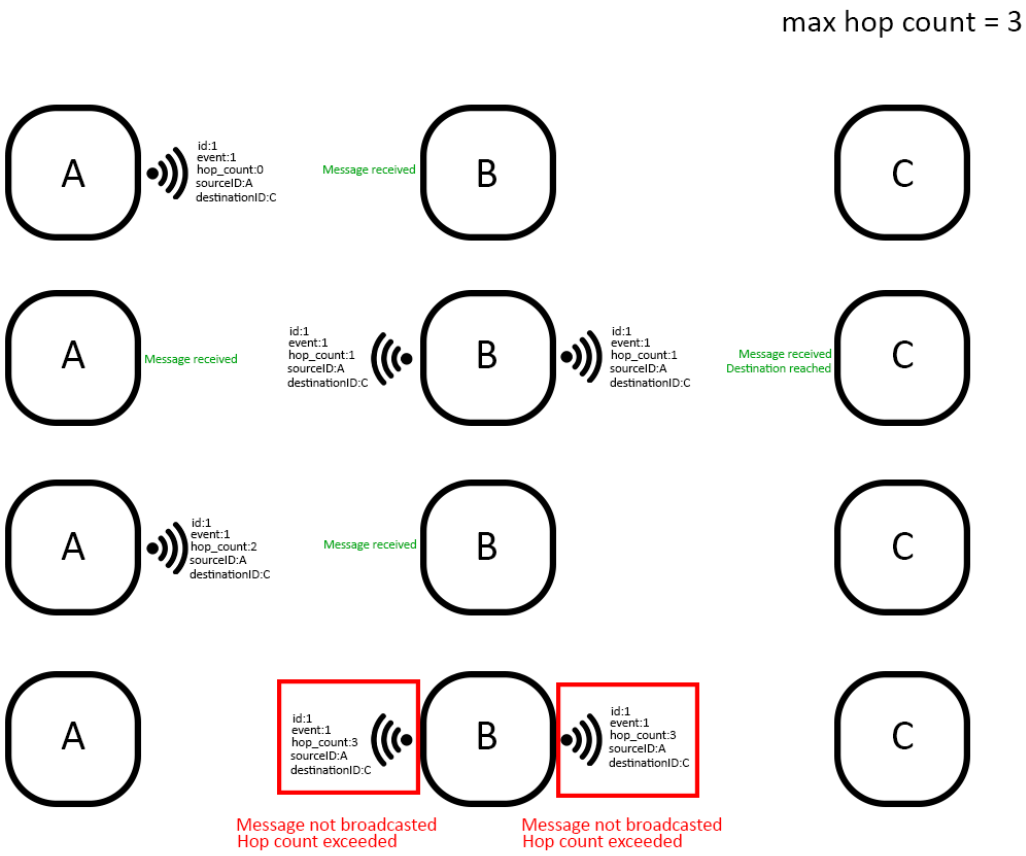

Figure 11: Pseudo-AODV in action over BLE and BLE beacons

Advantages of the super-sensor network:

The advantages of using such an Ad-hoc network are many. We can send updates to the model on every super-sensor nodes by making the controller node broadcast a special message containing the new model. The Ad-hoc nature of the protocol will ensure that every super-sensor will receive this update. The range of the super-sensor network could be theoretically extended infinitely by simply introducing new super-sensor nodes. All the nodes in the sensor network can be mobile and they need not have same neighbors all the time. The sensor network won't fail as long as every node has a path connecting it to the controller node.

Another benefit of this architecture is that sensors can talk to each other in this network. The controller node could have another reinforcement learning model to learn new actions by itself, thereby making the entire system a self-learning and self-functioning unit. Intelligence in the controller node will enable what's called 'Edge analytics', letting appliances become truly intelligent without relying on some cloud to dictate actions to perform.

We can deploy super-sensor hardware modules in buildings, and make smartphone applications also become a part of the super-sensor network by leveraging sensors on smartphone, making it act as a super sensor. The overall system will benefit by increasing accuracy in event detection happening through combined network of Smartphones and hardware modules.

Conclusion:

The project we implemented was able to function as a super sensor, detecting 2 events – Tap water turned on and microwave oven powered on. As previously explained, the learning models could detect infinite number of events once proper training data is provided. Our implementation used Magnetometer and Mic on Android smartphone, and gave a near perfect prediction accuracy. We also tested for dummy cases – We played sound of microwave over a speaker, and our application still classified it as background noise, since the Magnetic noise was absent from the readings. The proposed architecture is sturdy, and could be made to self-learn from new events using reinforcement learning.

The Pseudo-AODV protocol implemented works with the super sensors functionality to form a fault tolerant and connected network of super sensors. The protocol though implemented using JSONs still provides all functionalities of a standard AODV protocol. We tested 15 android devices connected via this Pseudo-AODV protocol and the network worked with latency not worse than that of a standard AODV implementation.

We created a network of 15 super-sensor nodes by installing the application of 15 android smartphones. This network was successfully able to detect every occurrence of the trained events (Microwave oven and tap water) happening within it's range.

Though the ‘super-sensors’ and Ad-hoc protocols have numerous ongoing and published studies, there has been no previous attempt to build a Ad-hoc network of such Machine learning based super sensors. Through this project, we have demonstrated that such super-sensor networks possess a great potential, and could completely replace traditional sensor systems (At-least for applications like home-automation). With possibility of deploying this system through smartphone applications, we could have an ubiquitous network infinitely scalable, that can detect every single event happening in the range of the network.

Scope for future work:

Though the project implementation is fully functioning, there is a lot more optimization possible. We have categorized the possible domains of work in following way-

1. Convert Pseudo AODV to a true AODV to ensure security, data compression and various QoS levels.

2. Train learning models on more sensors.

3. Increasing the prediction accuracy of the models by improving the learning model.

4. Create learning model that gives the same accuracy of prediction even if source of data point is at varying distance from the super-sensor.

5. Making every single node capable of reinforcement learning.

6. Modify the AODV to allow on-the-air deployment of updates for event classification models that the super-sensor nodes have. This will make it easier to add support for detecting increasing number of events in future.