Assignment 2 Report

15324730

Technical Diagram

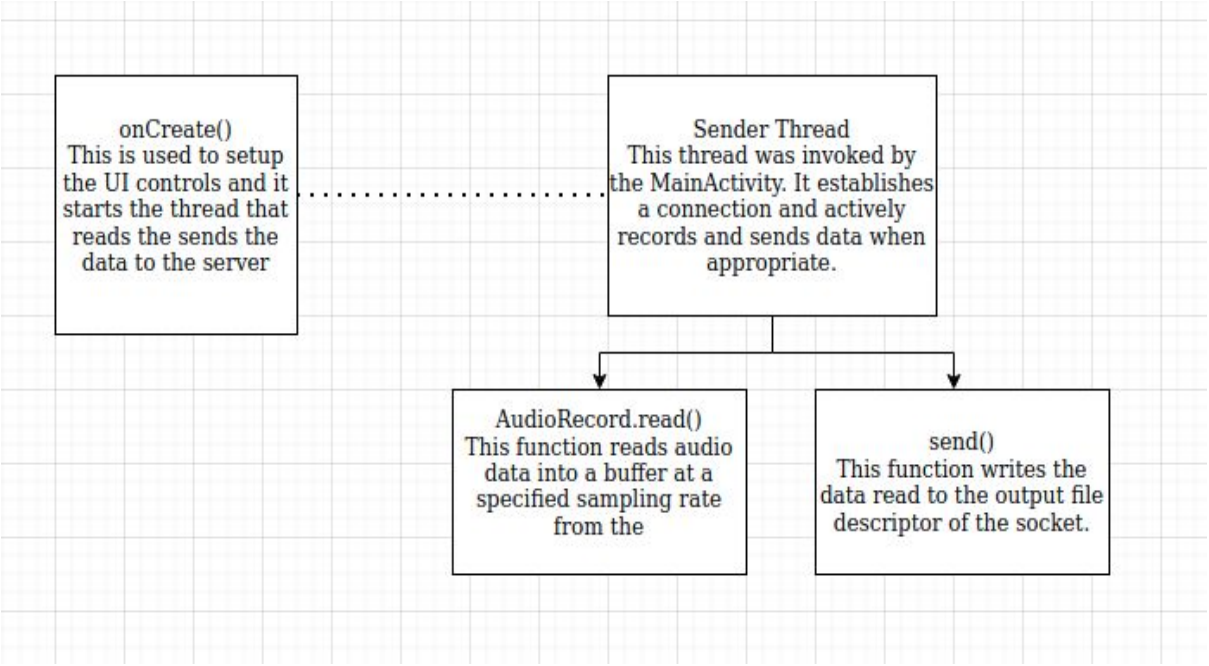

Fig 1.1 This shows the structure of the Android application.

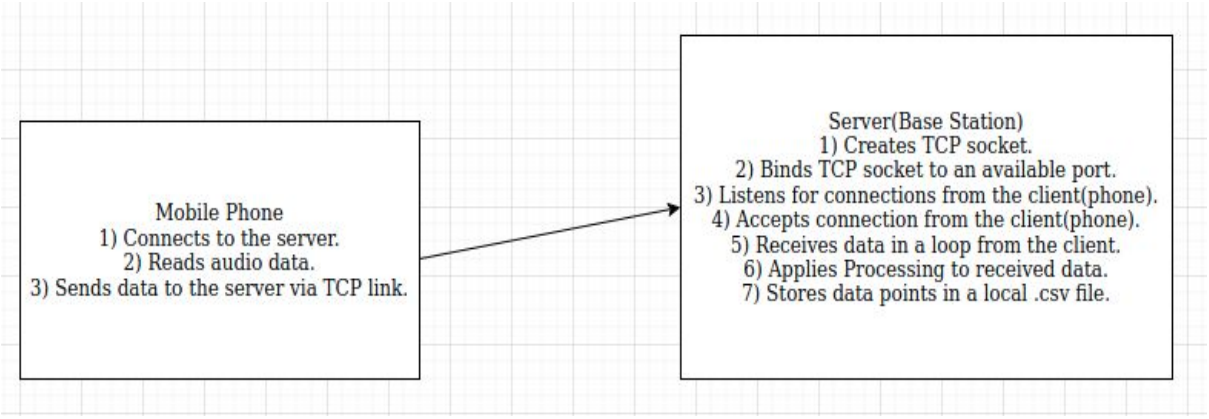

Fig 1.2 This is the client server model used in the application. The server runs on my laptop.

Collecting The Data

The data was collected from my main microphone on my android phone. The methodology I used to collect this data was to develop an Android application on Android Studios and a backend server on my computer.

The Android app I developed allows the user to start recording. When the app is launched, the OnCreate() method is called. OnCreate() is used to set up the user interface and to start the Sender Thread. The Sender creates a new socket that connects to the server.py script running on my laptop.

Once the user presses the start button, an onClickStart() function is invoked. This starts recording audio into a buffer at a sampling rate of 41000Hz. The buffer is made of short data types since the encoding on the audio recorder is natively PCM_16BIT. The encoding is also known as .wav encoding. This encoding is usually not used because it stores raw microphone amplitude values. This is not compressed and takes up a lot of storage space. However, this is perfect for estimating sound intensity on the decibel scale.

The sampled values in the buffer are sent to the server on my laptop that has already established a TCP link and is awaiting data to be received. Once the data is received, the values of the buffer are written into a .csv file.

Before the data is saved into the data.csv file, the amplitude values are converted into estimated sound intensity values that are then stored.

When the stop button is pressed, the app stops recording audio data and it stops sending the data across the TCP link. The start button can be pressed once more to restart the recording and transmission.

Discussion

A problem that I have encountered with the data is that the sensor(phone) is usually mobile and is not stationary. Therefore the phone must act as the client and the base station (computer) must act as the server. The problem is that the IP address of the might change if the phone moves location. This is due to entering new networks.

I would suggest adding a new thread in the app that checks the mobile phones global and local IP address by pinging a known DNS nameserver. If the IP address has changed, create a new TCP socket connection to the base station server. The server should maintain the state of all the connections. Maintaining the state of all active connections in the base station server may allow a remote control feature to be added. This will allow an operator in the backend to stop and start recording and transmitting data on all mobile devices from a centralised location. This may create a potential security risk as there is a single point of failure in the design with the onus on the centralised controller to be responsible. This could be mitigated by adding auditing logs and authenticated users on the base station system.

The current design does not feature any end to end encryption. In a further design I would add support for sending data/command messages via the HTTPS protocol. This would mean registering a domain name for the base station(s). I would then contact a certificate authority to issue a SSL certificate for the domain names. This leverages the X.509 root of security protocol for authentication of domain names. The certificate would provide a “legitimate” public key for the base station that could be used to set up a master key for synchronous end to end encryption which tends to be faster than public key cryptography.

If there is one base station in the system, this may produce availability problems if the base station goes down. This problem would have to be mitigated by providing replicated server clusters in multiple regions. The requirement of availability could be done by using Apache

Zookeeper. Zookeeper does not provide multi-region synchronous replication but it provides synchronous write replication and asynchronous read replication. This increases the read throughput of the system. However, the data that is read might be stale since the reads are read from all the followers and not just from the leader. This is in direct contrast RAFT which is similar to Zookeeper but it provides both strong consistency for both reads and writes. Zookeeper would suit better in this case(less expensive). If the base stations are run in Zookeeper servers, the availability of the base stations will be high and the read throughput if necessary in the future will be high. Load balancing will need to be applied to the system to make sure the mobile phones read/write to the right base station.

Privacy for clients should be provided by adding server-side encryption of the data. If there is a leak from the database(data.csv), the data will need to be unencrypted by the malicious actor. Anonymity can be provided by applying an audio filter that distorts the audio data without changing the mean sound intensity before the data is sent to the server. Other identifying tags like user-agent and precise location can either be omitted or obfuscated.

At the moment, my server is running on my laptop. My internet connection is a non-enterprise service. This means that the IP address of my server is not static because it is subject to the DHCP protocol of my local network gateway and my ISP. I would solve this by running my server on the AWS infrastructure where I can buy a static Elastic IP address for an EC2 instance. AWS can also be used to provide a distributed, fault tolerant and cheap database service that can be connected to my mobile app.

To incentivise users to enable the permissions needed(Internet and audio-record) to run the application, you could partner with a bigger company and add the application as a feature in their platform. Enabling the application could for example unlock free storage capacity in the cloud for the user. This is cheap and it allows the big platform to earn great public relations since they are seen to actively combat environmental pollution(noise pollution).

Conclusion

From an urban computing perspective gathering sound data from will participants and their phones might be a great way to gather highly granular data with high fidelity. To incentivize participants to activate audio-record permissions on their phone you would need to lump the app with a platform like Google Maps. This would provide Both location and sound intensity levels in an urban environment. The anonymised sound intensity data can be made available to the public. Researchers, county councils and urban planners can use the data to manage noise pollution and potentially implement solutions to increase the quality of life for the citizens of a given city. Being the stalwart of combating environmental pollution is an invaluable image in modern society, albeit the value is somewhat intangible.