

**SOFE 4790U: Distributed Systems – Fall 2020 (Dr. Q. Mahmoud)**

**Project Proposal  
Project Group 7**

**Location Tracking and Mapping System**

|  |  |
| --- | --- |
| **Student Names** | **Student Numbers** |
| **Marwa Safa** | **100585192** |
| **Walid Safi** | **100623815** |
| **Christopher Phan** | **100620028** |
| **Curtis Whall** | **100655399** |

**Table of contents**

[**Summary**](#_tfqtz3pcf8l7) **3**

[**Introduction**](#_9gd6zr2effwm) **4**

[**Background**](#_nwmj5kydzqkv) **5**

[**Related Work**](#_c9uvtjhh27cj) **5**

[**Location Tracking and Mapping System**](#_bn2j1xbrd8vv) **6**

[Proposed Solution](#_rb5r6g683woa) 6

[Tech Stack](#_61rpdu8frp75) 6

[Components and Technologies](#_dvr9bml7e1kd) 6

[Architecture](#_q5b1rlw1p8ua) 7

[Assumptions](#_r1cmjc8sae9w) 9

[Implementation](#_z2kio0izxh6o) 9

[Scheduling](#_979xetwp2pad) 11

[Replication](#_9lxl65yaaydq) 11

[Pinging](#_l89jmykjdjcm) 12

[Challenges and Solutions](#_buk4ihl26co5) 12

[**Evaluation and Results**](#_d59c7xfb88rm) **12**

[**Usability**](#_lju88a2rp0fo) **13**

[**Openness**](#_ry9i8o1imcse) **13**

[**Maintainability**](#_x47dm4xdpein) **13**

[**Scalability**](#_l9yoliekyixn) **14**

[**Reliability**](#_h07ij7xgqpet) **14**

[**Security**](#_m08rm0cscczx) **14**

[Failure Transparency](#_mv870qo1xwp3) 15

[Concurrency Transparency](#_8xvp3c5fnqk1) 15

[Replication Transparency](#_5qhdxruir4yr) 15

[**Conclusion and Future Work**](#_21cccdyq3j05) **15**

[Conclusion](#_8feekmofqv5i) 15

[Future Work](#_was2cl2dfwi7) 16

[**References**](#_v39ms873rpfe) **16**

# **Summary**

This project involves building a location tracking and mapping system. This has been accomplished by putting together all material learned in the Distributed System course. This includes implementing fault tolerance measures, such as mutual exclusion (Semaphores), heartbeat/pinging protocol using concurrent threads, and data replication for the client-side. We have used different technologies and designs for this project which are android studio, java-based server, google API, and firebase database. Android studio is used as the front end or client-side of the system. We have used a multithreaded client to continuously ping the server for updates, and Google maps API to create and map custom markers based on the longitude and latitude of the client. The backend of the system consists of a multi-threaded server that is responsible for scheduling and distributing data to clients. The read and write data is done over TCP/IP network sockets synchronously to the remote server. The location is also collected into a real-time database, firebase. It keeps updating every time the client uses the application, this can be used for real-time data analytics. The system is a centralized system architecture where one or more client nodes are directly connected to the central server and it is responsible for serving multiple clients. There are three main components which are the server or the master, nodes that are android mobile devices as clients, and the communication link which is TCP network sockets. Clients do not interact with other clients, and the server will handle co-ordination and messaging. All clients will receive synced data, if any client moves it will update the server and send the data to other clients’ devices. We were able to develop a functional location system while implementing methods and materials from class.

# **Introduction**

Communication with devices between people has become an essential part of human interaction over the past couple of decades. Being able to connect to other devices or objects over the internet is becoming more popular every year, more commonly known as the Internet of Things. And with the explosion of IoT (Internet of Things), communication between devices and humans will only increase, as of 2020 there are over 20 billion IoT devices and expected to almost quadruple to 75 billion devices by 2025 [3]. To fully utilize the vision of IoT, subsystems of devices are created, these systems are called Distributed systems. Distributed Systems are systems whose components or nodes are connected via network/internet (LAN, WAN) that “talk” with each other to accomplish their task. Currently, societies around the entire world would not be able to function without these systems, because of COVID-19 we are reliant on devices and the distribution of data over multiple different nodes. This includes, Google meets, School File Sharing systems (ex. Canvas, Blackboard), Online banking systems. The services these systems provide allow us to quickly obtain information through a distributed network that is hidden from an average user. Although not every distributed system is essential to the day-to-day lives of users, they are able to provide users with security, availability, and happiness. One of these systems that can provide security, information, and live communication between devices is a Location tracking system. To have information about your friends or family’s location during these times is essential. With the current state of the pandemic, people need to have a valid, real-time, and noninvasive location tracking and mapping system in place.

# **Background**

**Location Tracking**

In general terms, a location tracking application is an application that requires the user to allow their location to be published to certain sources and allow that source to track and/or apply certain conditions based on the user’s location. This location tracking system could result in positive and/or negative merits to the user.

The benefits of a location tracking system are that it is easier to be found in case of emergencies, theft recovery and increase in safety. By having a location tracking system, the users can feel secure by having all the family and friends tracking each other’s location. In case of any emergency, the first responders can find the user quicker by using this technology. There are some cons about having a location tracking system like malicious attack and the user information could get used and the attacker will know the user’s location. It is also a threat to privacy and the collecting of geolocation could be used in many ways against the user. The main focus of a location tracking is to increase safety but there will always be people trying to use technologies against its motive.

The functionalities of a location tracking are the tracking of users, seeing the exact location of the users and when they move, continuously updating, communicating with the users, sending directions and adding notes on how to get to a location. The location tracking system could have many functionalities added to the system by using AI to learn from users like the place most visited, add home location for each user, add work location for each user and many more functions could be implemented.

# **Related Work**

There are many location tracking systems in the market right now. We will discuss a few location tracking systems below and how it works.

* **Life360:** it is an information technology company that provides location based services and taimed at a family social network where all the family members can see each other's current location. It also provides notifications, driving reports of users, and directions from one person to another. []
* **Google Maps:** Web-based Service hosted by google that provides information about the area. This application allows users to find certain stores, routes, and areas. This provides directions with oncoming traffic warnings and also has notifications.Does not have location sharing and does not have any sort of security. [ ]
* **Famisafe:** A Parental control mobile application that enables real time location tracking and device limitations and they have monthly fee for using the services. A unique aspect of this application is the device limitation where the parents are able to monitor the child’s phone as well. []
* **Find my Phone:** Exclusive IOS phone tracking device. [ ]

# **Location Tracking and Mapping System**

## **Proposed Solution**

Our proposed solution is to create a distributed system that can accurately obtain a device's location and send data of every connected device name, longitude, and latitude to every connected client. Once a node has all the locations of the other devices, we will display the information on the map via a marker, and the name of the device's owner at that location. To be able to develop this we had to define the interactions between our nodes, what programming languages and protocols are the best to implement. Types of data structures, and features to be implemented.

## **Tech Stack**

* Android Studio
* Java Server
* Google Maps API
* Firebase Database

## **Components and Technologies**

We considered using multiple different devices, this included a computer and web interface frontend, laptops. However, the mobility of these devices, especially during these times did not make them a strong choice for a location mapping system. We decided to create the client-side using mobile applications. This was by far the most viable, and the most logical solution, since almost every person has a mobile device and is carried with the user at all times, and usability for the client is convenient. In Combination with a mobile application as a client, we created a Java backend server, the main reason we chose java is that our group already has previous experience using java server architecture. Aswell, Android Studio operates with Java, which will reduce the complexity of connecting the server and client.

To create a connection between the nodes, TCP/IP socket protocols were used. The Java server will listen on a part waiting for clients to connect, once connected we initialize a data stream. Although java RMI and ZeroMQ seem to provide an asynchronous message system, which would have made communication between nodes easier, these frameworks and methods are not supported on android studio. Using TCP sockets we were able to establish a connection between the client and server, and create Object input streams.

As we used the Android studio development tool, this granted us access to use Google Maps API. The API and android studio are already compatible as Android studio has implemented google maps internally. This meant the framework for implementing the API with the android studio has already been documented and supported.

Firebase Real-time Database, We created a connection to a cloud-hosted database. This database stores data using JSON and is synchronized with every client it is connected with. It will allow our applications to receive real-time analytical data based on the server, or other clients to provide local functionality. This may include graphs, client count, etc. Although not implemented fully, we have established a working connection as well as adding data to the database.

The following diagram shows a simple interaction and flow of data between a node (Mobile Application) and the Backend Java server.

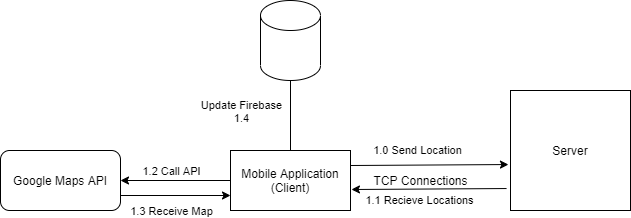


Figure 1.0: Client Data Flow Diagram

## **Architecture**

Our Distributed systems architecture is a Client-Server Architecture, however as we servicing more than one client, it will be considered a centralized System architecture. One server will be managing data streams between multiple clients. As we use TCP, the server is able to handle a large number of connections at once.

Developing the system using a centralized architecture was chosen as this architecture represents the proposed functionality the best. This architecture includes 3 main components, the Node (Mobile), the Master (Server), and the connection link. These components were described in the previous sections of the report. Decentralized system architecture for this system is not a viable solution, as every device that needs to be connected must communicate with each other. Acting more as a peer-peer system. Not only will this involve a high amount of network traffic between the nodes, but it will also be prone to data corruption. Each node can act as a point of failure, negatively affecting the other nodes. However, using a centralized system architecture the data processing component is left to the main server, which will have much more dedicated resources to handle failures, synchronization, etc.

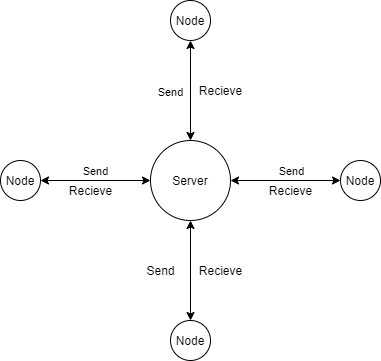


Figure 2.0: Centralized System Architecture

There are many benefits of using a Centralized System Architecture for a Location Tracking and mapping system. Clients do not need to have dedicated hardware to be able to open the application. Not all android devices are the same, removing the data processing from the client provides the client with lightweight applications that may not have been able to run on the device. As Well as using a real-time external database like Firebase, the client does not need a local database like SQLite. The centralized architecture can quickly add, and remove a node from the system, as well as quickly receive and send data to access nodes. If a phone is no longer active, the server will remove that node. Nodes are also able to independently, if one node fails the server is able to handle that failure without costing the performance of other nodes, this is cost-effective especially for smaller systems. The main advantage and the main reason this architecture was chosen is that the server is able to act as a global clock to the clients. This means it is able to synchronize all the data received at the time and return the same dataset to every client, such that there are no discrepancies between the nodes. This makes it easier to achieve consensus as only one node (Master) controls it. Also, having a centralized server ensures that potential data corruption is corrected before being sent.

Although the Centralized system Architecture is the best suited for the proposed system, it is still susceptible to failures. This architecture has a few disadvantages that could negatively affect the performance of our system. These can range from a Fail-stop failure to a fail-safe failure. As this architecture is extremely reliant on the network connection, the system can fail when the central node shuts down, loses connection or exhibits data corruption. Unless measures are taken to reduce the central node's potential issues (By implementing Fault Tolerance methods), it may become very costly to maintain.

## **Assumptions**

During Development the following assumptions were made:

1. The Connection between is Client and Server is secure
2. The Devices that will use the system will be a mobile device with a minimum Android SDK version of 24
3. Android Device will provide valid longitude and latitude values whilst using the location manager provided by android studio
4. Internet connection on the server and client-side are secure and will not disconnect

## **Implementation**

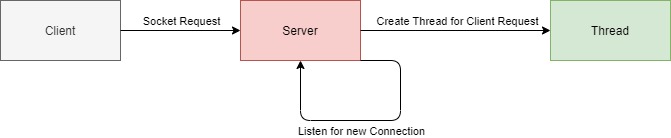
The server was developed using Java, and TCP sockets. However, to be able to fully accommodate the size of the project. We had to develop the server by using a multi-threaded socket connection. The server continuously listens on a port, in our case port 5556 waiting for a connection request. Once a connection is detected it will accept the connection and create a new thread, this thread is given another port number if there is already an existing socket connection on the original port. This was implemented by using an infinite loop to wait for a connection, and a thread Handler class, to initialize classes, create an object input and output stream, and copy local data. Once a thread has been initialized it is able to access the server resources, this includes the local data saved on the server, as well as the computation and data processing functions. The thread then returns an object through the streams created earlier to the corresponding client that is listening to the port number assigned when accepted.

Figure 3.0. Multi-threaded Server Diagram

The Server will create a local list, this will become a local database to increase the reliability and security of the system. The server would add the new client information, after checking if the client already exists. If it exists the server would update the clients’ longitude and latitude and return the list of locations. The server will also check if the client id (name) is valid, if the name is null, or does not have any characters, it will not be added to the locations list as it will be a random marker on the map.

The Server will also implement server-side logging. If the server crashes it stores the locations and people who have connected in a file. This file can be used to restore the state of the server if some clients are no longer active.

The Client was created by implementing a google maps activity from android studio. To request a socket connection we created a thread in the server to initiate the request in the background. The client would require an IP and a port number of the server. The client would then send a request to that address and port number and wait for an accept() response or timeout. As android studio does not allow socket connection on the main process, we created a button to start the thread and the connection, this would allow the system to easily create a connection whenever it is needed to access the server, without straining the system with infinite calls. The client would receive its current location using a location manager class. This class would use the android internal server called LOCATION\_SERVICE, and well as FINE\_LOCATION. Continuously calling the following two functions to receive the client’s longitude and latitude.



Figure 4.0: Location calls for Clent

The client would store these values into an array list, and send the information to the server. The server would respond with a larger list containing its own location, as well as the location of all the other clients that have requested the server. The following images are showing the functions used to receive the server response and store in a local list. As Well as creating a marker object, and displaying the object on the map.

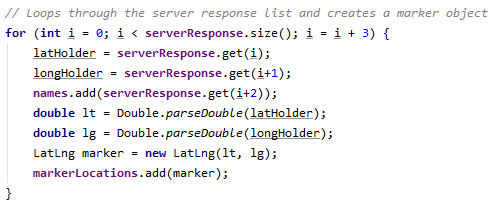


Figure 5.0: Receiving server Response

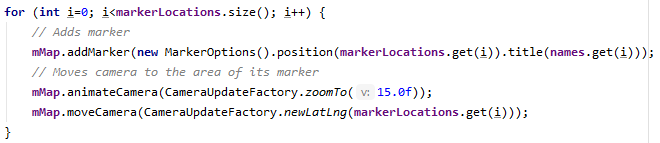


Figure 6.0 Creating Location Marker

## **Scheduling**

Our system consists of many different connections and synchronization. To combat the problem of corruption with the data. We would implement a semaphore into the system, a semaphore is a distributed mutual exclusion that helps protect key resources. To implement a semaphore on the system, we applied some functions; these functions consist of Enter(), Access(), and Exit(). A semaphore requires and creates a way for mutual exclusion to be safe and doesn't crash(ie. Deadlocks, thread starvation).

In our system, we implemented a solution to the race condition with multiple clients and also applied locks through semaphores to lock critical sections which allow for the desired client to access the resource and input data into the resource without any corruption. This would satisfy a fault tolerance where if the server crashes due to user inputs, then the client would still be able to use the app.

## **Replication**

Our system implements a strategy learned during the distributed systems class. We decided to replicate the server data every time the client requests information. Instead of reading the server information, then deleting it. The client saves the information and creates a save state mechanism for itself.

In the case of a server failure, the client is able to store the last known locations of every client connected to the server. Once the server is back online, the client will request the information once again.

## **Pinging**

To ensure that the client has up to date data, we created another thread on the client-side. The responsibility of this thread is to continuously call the server, updating its information and updating the client’s local storage. This is being done concurrently with the application, thus the application will not notice this happening and can run local functionalities without interruptions.

## Challenges and Solutions

|  |  |
| --- | --- |
| **Challenges** | **Solutions** |
| Creating a Socket Connection on android Studio was challenging as a connection is not allowed to be created on the main thread | Utilized multithreading android to create a connection whenever needed |
| Creating a Heartbeat and refreshing the map module every ping | Unable to solve this problem as we are unable to call the mapping function outside of the main process. Worked around by using a button to manually refresh |
| Connectivity over a network was difficult to implement using sockets | Port Forwarded and opened ports on the router and system, the opened ports and IP were able to work on the host’s system |
| Developing a function of the server to restart when data corruption or timeout occurs | Using System and execs classes, the server will execute code, however, it would not rerun on the cmd line. |

# **Evaluation and Results**

Throughout the testing and development process, we would consider the different performance measures we wanted to reach. We tested amongst these criteria: usability, openness, maintainability, scalability, reliability, and security.

# **Usability**

To evaluate the usability of the system we mostly just tested the basic features of the system. To test this, as we added features and components to the software we would ensure that it was meeting the requirements we originally set out. An example of this would be as we implemented the feature of using the GPS location of the phone to get a location to display, we would move the phone’s location, to ensure that it would properly simulate a user physically moving with their phone to another location. Another example would be ensuring that other clients could be seen on a client’s map, showing that the core functionality of being able to see other user’s locations is correctly functioning. These tests can be seen in the video provided.

# **Openness**

The openness of the system was also considered while developing and afterward. Due to the nature of the product, new features would be easy and cost-effective additions to the current system. The system’s current setup would allow for threads and or multiple views and can contain more data about the client if needed. Additions such as timestamps of when the user’s last location update was, any restaurant recommendations nearby, and similar features could be added with minimal impact to the core functionality already imputed. The system only requires a location update every 15-20 minutes in a real-world application, so there would only be stress on the client during these times of sending and receiving, and would more or less be free to complete other tasks during the downtime. Overall, we believe that our current system is a perfect platform to have further features implemented.

# **Maintainability**

Looking at the maintainability in our system we can conclude that maintainability is an area that could be improved. Currently, the software is running off of one server that would need maintenance to ensure that the system continues to function properly. There is an upside with our system when looking at the costs of maintaining, which is that the nodes/clients in our case are the smartphones of our users. These users will most likely upgrade their own phones over the years which would be a cost we would save. There would still exist the cost to maintain the server(s) however. The use of java would also help in this sense with the abundance of programmers that know java and can maintain the code. This section was hard to test but based on practices followed and previous knowledge this product should be easily maintainable.

# **Scalability**

As our user count increases so will the load on our servers. For testing processes, we only used one server, but this could be improved with the actual release of the product. The addition of more servers was not tested directly but there was discussion of what it would look like if this implementation occurred. Most likely, we would add servers to increase user count in an area, and then have these servers communicate with each other, providing other user’s information. Along with a future update to allow you to create a group of friends or family, these groups would have a maximum limit on a server, reducing the strain and limiting it to a certain threshold for a server. We did a test with a maximum of 4 emulators making calls every second to test the load and strain on the server, and the server performance did not suffer. These calls would occur much more frequently than normal, and can be seen as a test for more clients. (It's hard to test with phone emulators as they require more resources than normal programs and calls.

# **Reliability**

From the testing that we completed, our system is overall reliable. With the server and clients left online for roughly a day to run, there were no issues to be reported. The server maintained connectivity and the clients continuously updated the server. While this seems like a small scale test, without the acquisition of a large scale server and reliable internet connection, this was the best that we could do. We also tested the reliability of the clients without the server, while the functionality will no longer work after the server goes offline, the client keeps the data saved to continue to display locations. When the server comes back online, the clients connect, send and receive data like nothing ever happened.

# **Security**

This was the least focused on evaluation and testing that we have done. Security is an important metric in our system, as it deals with the location of the users, as well as other personal information added in future updates. Currently there is no form of security in the product, but there is room for the addition of encryption of data and protection from outside attackers. This is a major field we need to address in coming versions of the product.

## **Failure Transparency**

The Client is able to run even if the server crashes. The client is unaware of the crash as it uses the local repository of locations before it pings. Even if it pings the server it wont update, or send a message to the client. The client is still able to complete their functionality despite the failure of the software components. This was tested by shutting down the server while multiple clients were running. The clients were able to interact with their local functions, buttons, text fields without any error. This completed mainly because of using concurrent threads.

## **Concurrency Transparency**

Our system is able to achieve concurrency transparency by having multiple client processes access the shared resource the server side. This is further enhanced as we implemented the semaphore locking mechanism. Clients are also able to concurrently call the server in the background via threads, accessing client side information, and allowing the mobile applications to run without interference. The server concurrency was tested using 1 second interval calls, with 4 different emulators. They successfully accessed the shared resource without any errors.

## **Replication Transparency**

Our system achieves Replication transparency as the system will create a local copy on the client side without the users knowledge. If the server crashes the client is able to access the last known location on their own device. This replication is updated every time the server is pinged. This was tested by shutting the server down and refreshing the client, the client still displayed the locations of last known location.

# **Conclusion and Future Work**

## **Conclusion**

For our project, we have completed a fully functional application of location tracking and mapping system that is able to take user’s input of their location as well as mark user’s current location from multiple clients and also store the data on a real-time database accurately and efficiently. We are able to put our knowledge of the entire course to good use and increase usability, reliability, availability, maintainability, and scalability with this project because we have implemented a wide variety of key points mentioned and taught in class. We also applied other classes’ techniques to this project in order to execute our goal and develop this application. Some functions that we implemented with the knowledge and resource from the class would be mutual exclusion, multithreaded system, socket programming, and IP networking and/or calls.

With the creation of the project and with great research for this project, we learned a more in-depth technique on the implementation of distributed systems and its interaction with other key components not related to distributed systems.

## **Future Work**

Although our application is fully functioning and meets the requirements that we planned, there are many ways to patent and improve our application. Some ideas of future goals and work in progress create a broker to handle scalability at a better rate by assigning groups of devices to a server more suitable for their size, implementing a security system in order to provide users’ with their own privacy and satisfy the privacy requirements, this, in turn, allows increases security, and to further develop our Firebase database to provide the users with a real-time data analysis which would be quicker and more accurate in terms of location mapping. By implementing a security system, this would not only provide the users with their own privacy and satisfy the privacy requirements but it would also make our application a more reliable application where everyone would enjoy with their own comfort, people would be able to be less stressed about their breach of privacy. We would also like to implement and improve on the application’s overall UI and performance compared to the demo shown during the presentation in order to have a solid application worthy of publishing.

# **References**

[1] *Google*. [Online]. Available: https://developers.google.com/maps/documentation/android-sdk/start.

[Accessed: 08-Dec-2020].

[2] “LocationManager : Android Developers,” *Android Developers*. [Online]. Available:

https://developer.android.com/reference/android/location/LocationManager. [Accessed:

08-Dec-2020].

[3] M. Vega, “2020's Internet of Things Statistics, Facts & Predictions,” *Review42*, 21-Nov-2020.

[Online]. Available: https://review42.com/internet-of-things-stats/. [Accessed: 08-Dec-2020].

[4] Parikshit HoodaCheck out this Author's contributed articles., P. Hooda, and Check out this Author's

contributed articles., “Comparison - Centralized, Decentralized and Distributed Systems,”

*GeeksforGeeks*, 30-Apr-2019. [Online]. Available:

https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems/.

[Accessed: 08-Dec-2020].

[5] “International,” *Life360*. [Online]. Available: https://www.life360.com/intl/. [Accessed:

08-Dec-2020].

[6] *Google*. [Online]. Available: https://www.google.ca/maps/about/. [Accessed: 08-Dec-2020].

[7] “The Most Reliable Parental Control App,” *FamiSafe*. [Online]. Available:

https://famisafe.wondershare.com/. [Accessed: 08-Dec-2020].

[8] “Support,” *Find My - Official Apple Support*. [Online]. Available:

https://support.apple.com/en-ca/explore/find-my. [Accessed: 08-Dec-2020].