FriendCluster

Matthew Busch, Jason Lee, Nathan Schwerdfeger, Lucille Wang

Motivation/Problem

Keeping a Group Together

- Trying to keep track of other people or groups in a large and/or crowded area is an issue that everyone faces day to day.
- Examples:
  - Staying together at an amusement park such as Disney or Six-Flags
  - Keeping your child near you
  - Meeting up together for a meeting/hangout
  - Making sure no one leaves a designated area
  - Busy streets in big cities such as NYC
  - Teachers keeping track of students on a field trip
  - Ensuring group safety in all of these situations

Solution

Solution: FriendCluster

- FriendCluster is a web-based application that utilizes geolocation to help a group meetup and to notify the group when a member deviates from the group.
- Our application utilizes localization and applies dynamic spatial alarms to adapt to mobile users.
- Deviation results in an alarm to alert friends of a possible issue, creating a safety network

Architecture

System Architecture

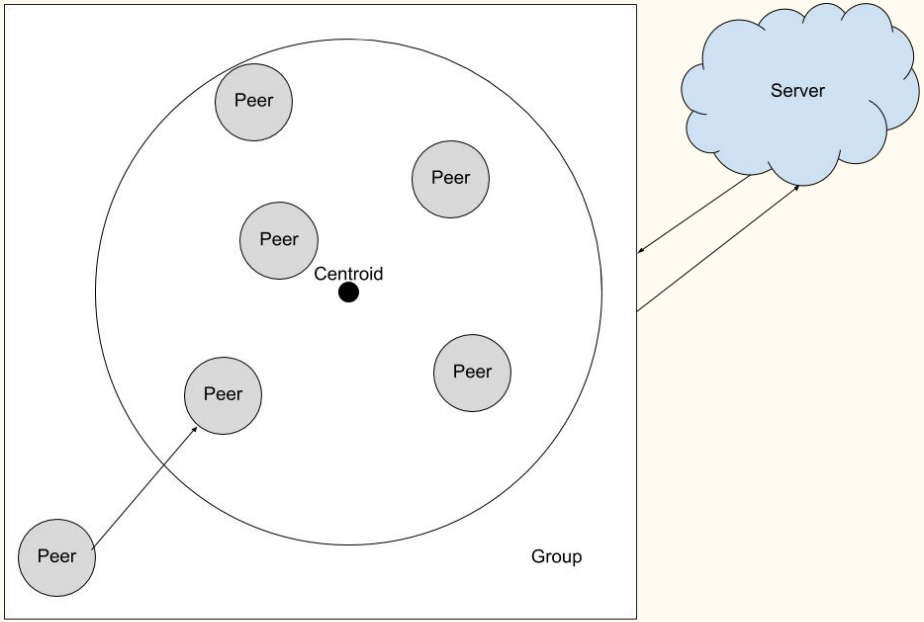


Figure 1. Architectural Design of FriendCluster

- Peers are a part of a cluster.
- Peers can join and leave whenever during tracking.
- The location of all the peers gets sent back to the server, which is where the source code lives, and determines the centroid and the safe zone depicted by the dot and the circle in the diagram, respectively.

Methodology and Approach

Inspiration - GT Mobisim

Pros:

- Inspiration for visualizing complicated traces
- Inspired many of the features present in our version of simulating traces
    - The usage of the term delay was an example of one of these features we decided to adopt from GT Mobisim.

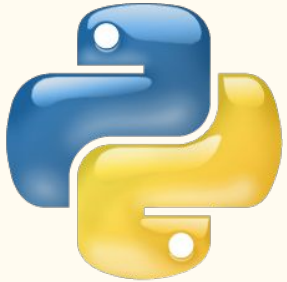
Cons:

- Over-engineered for our project
    - GT Mobisim simulates roads where FriendCluster needed to simulate all sorts of areas including parks.
- We needed a simulation program with a modular design.

Technologies Used for Simulation

Python:

- Language used for the source code
- We chose to use python because it allowed us to complete a rapid development of our prototype.
- Python provided us with many libraries that fit our needs.



Matplotlib:

- This library was used to provide us with the ability to visualize our traces and simulation.

Technologies Used for Backend Development

Node.js
- Run javascript on a server
- Great open source libraries

Express
- Simple, unopinionated server framework

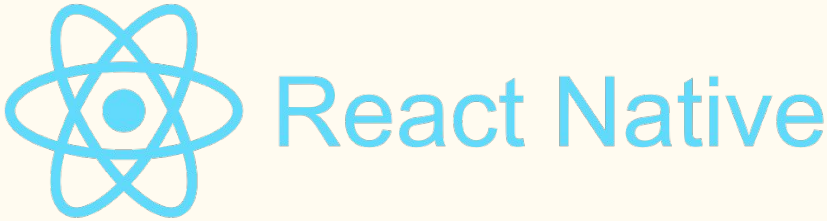
MongoDB Atlas
- Cloud, database-as-a-service

Mongoose
- Object Data Modeling library for MongoDB

Technologies Used for Frontend Development

React Native

- Cross-platform deployment
- Keeping javascript on the frontend and backend, reducing "context-switch" time



“Safe zones”

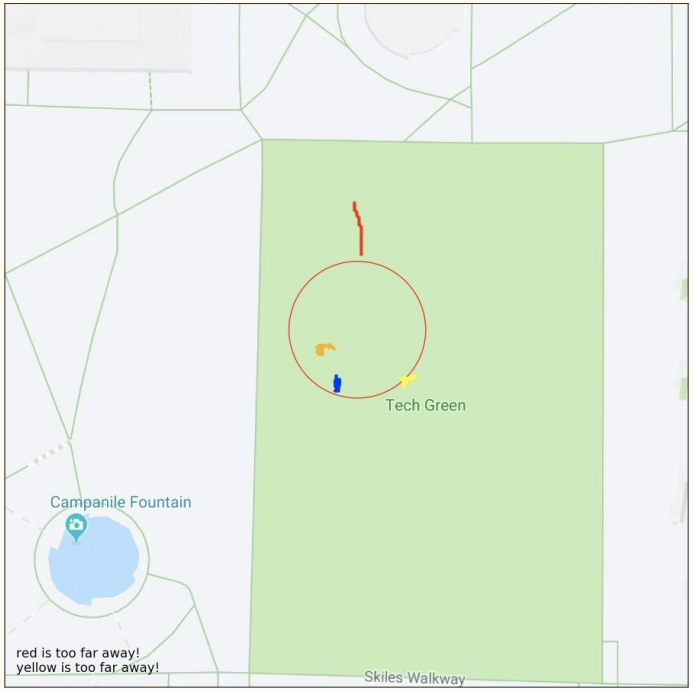
- Area determined by the backend in the server that is considered within acceptable closeness to group
  - Deviating from this area triggers an alert
- 3 Types of Safe Zones:
  - Should be able to either have a dynamically calculated centroid circle
    - Ex: Friends trying to stay together at a park or music festival
  - Circle centered on a single person
    - Ex: Mom and child
  - Circle fixed on GPS location
    - Ex: Don’t leave the designated or rendezvous point
    - Ex: Meeting up for a group meeting

Walk/Trace Generation

- For development and testing, we created a random walk/trace generator
- Inputs:
  - Heading - 0° to 360°
    - 0° is North and proceeds counterclockwise
  - Strength - 0 to 100
    - 0 creates a random walk
    - 100 follows the heading exactly
    - Strengths in between probabilistically follow the heading
  - Delay - integer representing number of frames
    - the weight parameters are ignored for the first n ticks
- Output:
  - Text file containing the coordinates of each frame of the random walk
  - Each text file simulates one moving object

Walk/Trace Visualizer with Simulation

- We used the Matplotlib library to help us create our trace.
- This included visualization of the peers/agents along with the “safe zone.”
- Safe zone calculation is determined by the centroid calculation.
- The visual also provides a notification in the bottom left corner of the window, as seen in the image to the right, if a peer leaves the “safe zone.”



Centroid Calculation

```
def get_centroid(x, y):
    return (sum(x) / len(x), sum(y) / len(y))
```

The centroid calculation is just simply the average of the x and y.

```
def in_circle(cx, cy, r, zipped):
    return [(z[0] - cx) ** 2 + (z[1] - cy) ** 2 < r ** 2 for z in zipped]
```

Using the centroid, we can figure out if the user is in the “safe zone” or not.

1. Calculate the distance between the centroid and the peer.
2. If the distance is less than the radius of the circle, the peer is inside the “safe zone.” If the distance is greater than the radius of the circle, the peer is outside of the “safe zone.”

Server Details

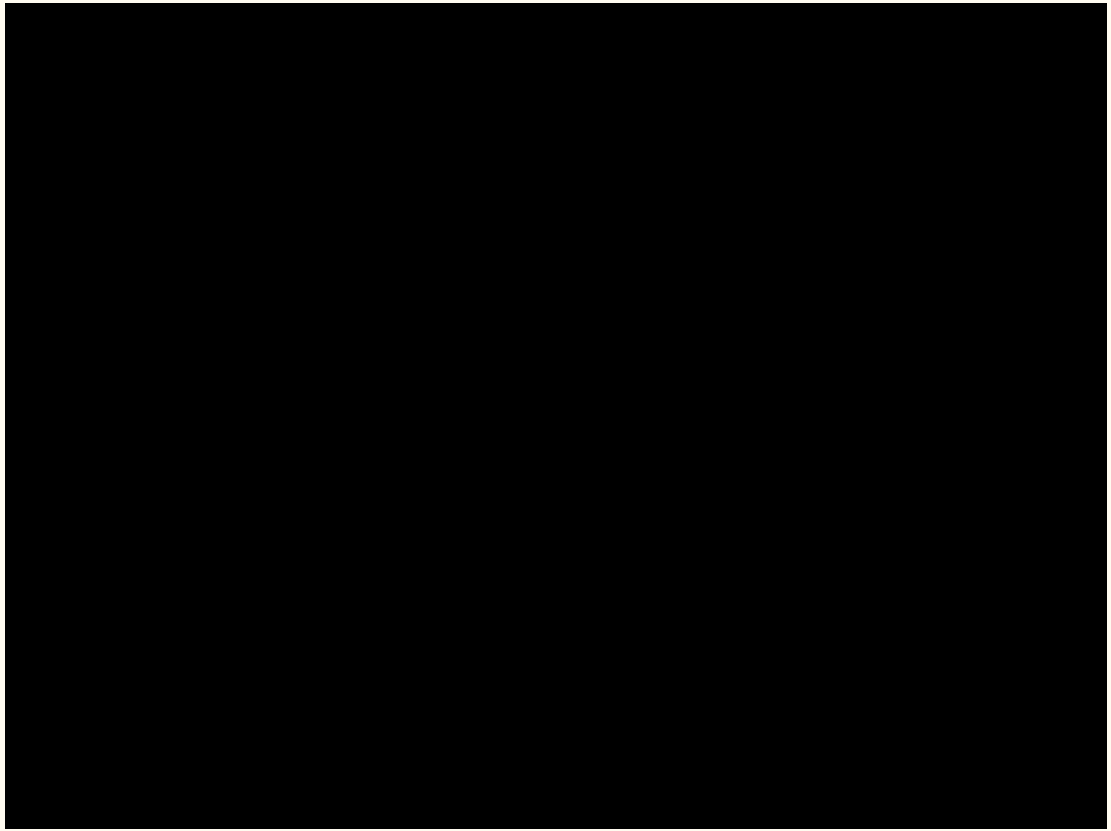
- The server is configured to accept basic CRUD functionality
- Group functionality is not yet added but can easily be extended
- Users can be created, updated with current location, and queried
- The centroid of the circle involving the current user can be requested
- The client sends an update of current location every 10 seconds and then asks for the location of the updated centroid

Demonstration

Simulation - Small Group, Stay Together

- This scenario uses our simulation to show a group of three users walking together.
- The circle indicates the safe zone, and when the users move, the safe zone adjusts and moves with them.
- The circle is always centered around the centroid of the group .

Simulation - Large Group, Stay Together

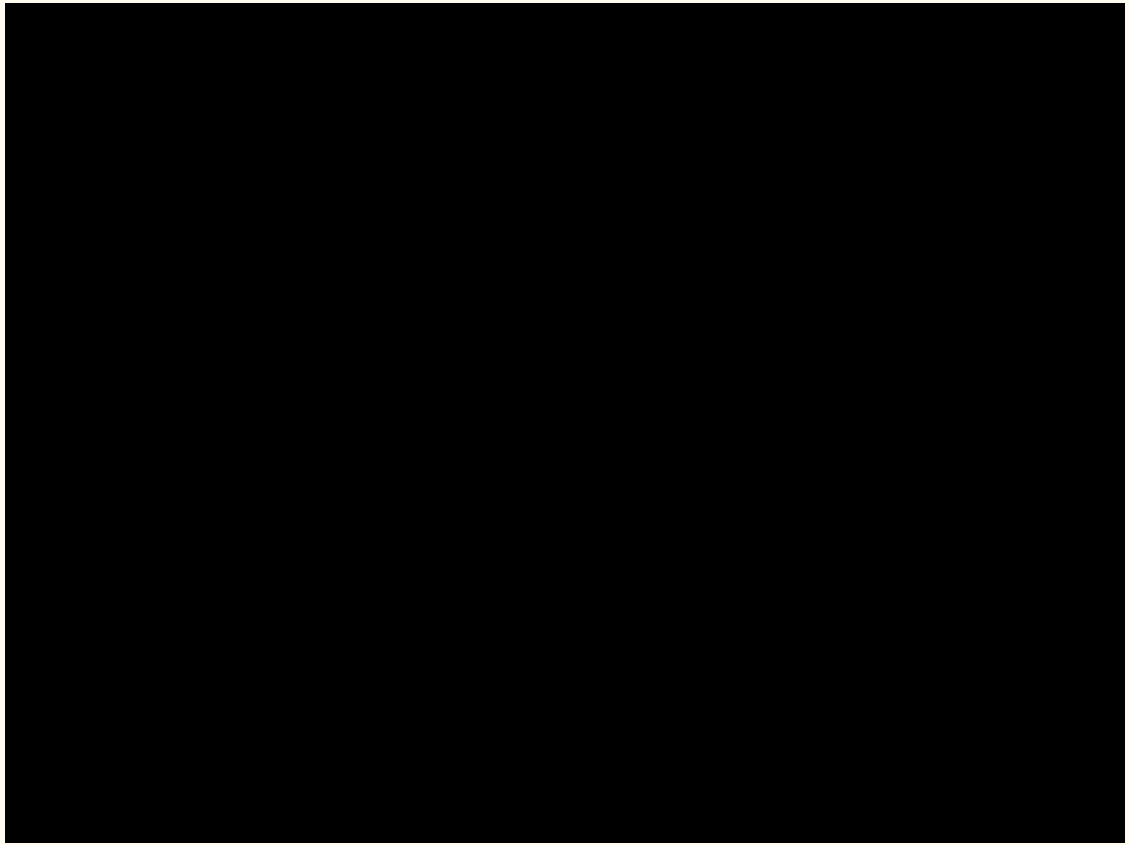


- This scenario uses our simulation to show 8 users moving together.
- With the increase of size, the centroid is still quickly calculated and the circle is found.
- Even with an increase in users, (tested up to 20), there was little latency or delay in calculating the circle, or in calculating users out of bounds.

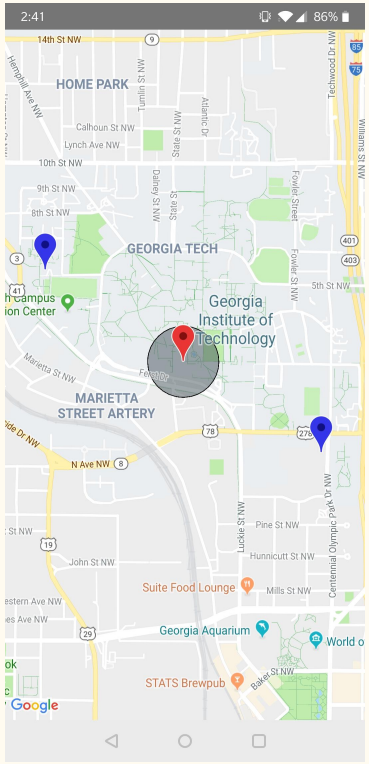
Simulation - Member leaves and comes back

- This scenario uses our simulation to show 4 users and a single user moving away from the group.
- When the red user leaves the circle, the circle becomes red indicating someone is too far away, and the out of bounds user is identified in the corner.
- The yellow user also briefly leaves the safe zone, and the simulation identifies both users as too far out of the circle.
- This indicates that the application we build will be able to send notifications when a user leaves the safe zone.

Simulation - Members meet up



- This scenario uses our simulation 4 users meeting up.
- This is an alternative use for our application where users select a fixed zone that users need to stay within.
- The safe zone size can be adjusted, and when users are outside of the set zone, the other users are alerted.
- When all users enter the safe zone, the alerts disappear.



Mobile prototype showing calculated centroid and circle along with two users (in blue)

GET http://nlab.fortiddns.com:5675/users

Pretty Raw Preview JSON


```
{
    "users": [
        {
            "location": {
                "coordinates": [
                    33.77053235,
                    -84.39242808
                ],
                "type": "Point"
            },
            "_id": "5cb585d047309b456c984553",
            "id": "ebeae856bf250538",
            "__v": 0
        },
        {
            "location": {
                "coordinates": [
                    33.77746137,
                    -84.40498699
                ],
                "type": "Point"
            },
            "_id": "5cb771111d512f4908ae3396",
            "id": "6de92d77799b27ec",
            "__v": 0
        }
    ]
}
```


Server responses in JSON format returning user data with locations

Mobile App Demo

- The mobile app prototype is capable of following the server protocol described previously
- The client sends an update every 10 seconds and updates its display with the circle centered on the centroid sent by the server
- Further improvements will add grouping capabilities for users (instead of using all registered users)
- Circle will be customizable in radius and behavior

Extensions

Prototype

- Our goal for this project was to create a prototype of our project to provide a proof of concept which can be later expanded on for a full scale project.
- Through our simple server and our simulations, we were able to simulate a basic version of what our end product would be.
- Also, by creating a prototype and proof of concept, we are able to show use cases and how our application could be used in the real world.

Future Work

Looking forward, future work that need be completed to bring our application to deployment include:

- Traces from GPS logging application
  - This will get one step closer to the deployment scenario because in the real world, the locations will not be coordinate points, but GPS coordinates
  - This will also allow us to discover any bugs and make the platform more robust
- Cloud-based backend deployment
  - This involves moving the centroid and safe zone calculation off the client device. We have a very simple form of this working.
  - The final cloud infrastructure must be scalable to allow for heavy loads.
  - This also enables multi-user real-time reporting.
- Mobile app implementation
  - The mobile app implementation will be supported by the cloud and available for both Android and iOS.

Future Work (cont'd)

- Mobile app implementation (cont'd)
  - Users will install the application on their phone to be able to use their own phones as the peers and communicate with one another.
  - This will also allow for testing of the system in the real world.
- Address any privacy and security issues
  - This would involve when location could be pulled.
    - For example, location can be only pulled when the application is running
  - How long keeping location data in our database is acceptable
  - Making sure our application is not exploitable since location could be considered sensitive data
- Platform Optimization
  - Apply best practices from spatial alarms and research from spatial alarms.
    - Ex. the energy and accuracy tradeoff. Find a good balance by throttling the rate at which information is being sent to and from the server depending on how close a user is to the boundaries of the "safe zone"
    - Implementing velocity dependent reporting

Extra Features to be Added

After development of our application, the following extra features can be added:

- Modify group after formation
  - Currently, the groups are fixed once they are made. Being able to modify groups after formation means that someone can join the group if they arrive to the event late or leave the group if they must leave the event early.
- Look for nearby groups/friends
  - This will allow people to walk up and join and see what groups are going on around them.
  - Also, an individual can see if any of their friends are closeby.
- Social Network
  - This allows for quicker development of groups.
  - Allow for integration of the friends of friends network.
  - Integrate login with Facebook/Google to connect with people already in your social circle

Results

Mobile Location Accuracy with JavaScript

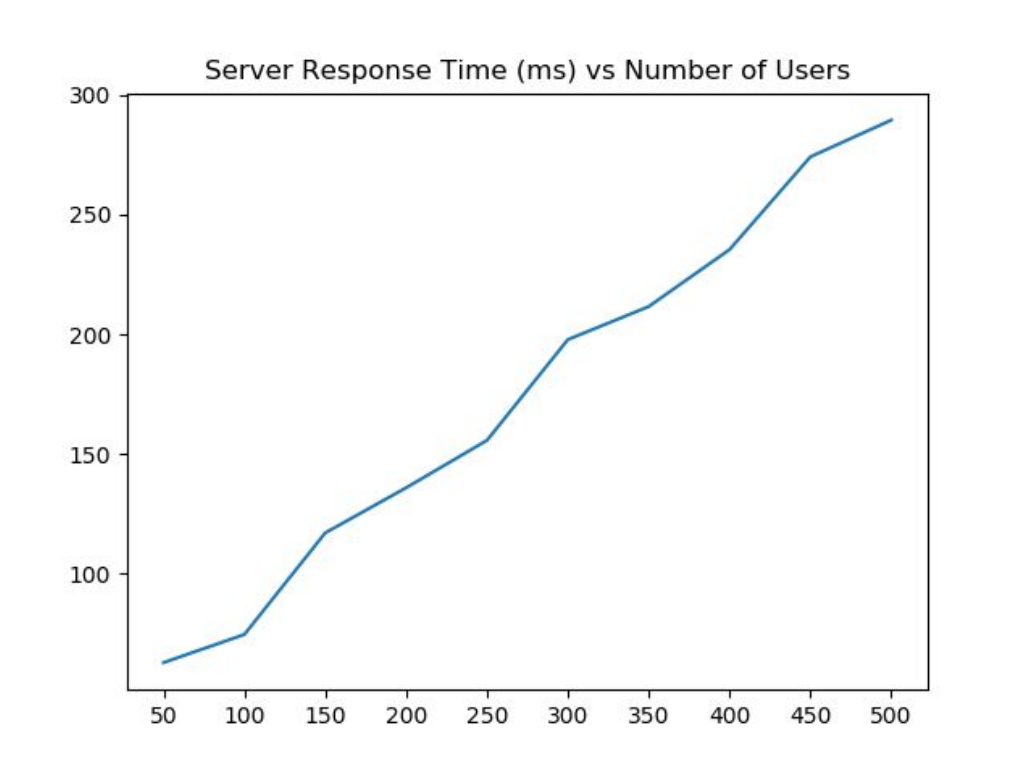
- Using the built in JavaScript geolocator module, location accuracy is quite poor

Measured Location	Actual Location
33.77746137, -84.40498699	33.777227, -84.396500
33.77053235, -84.39242808	33.770689, -84.391802
33.77043994, -84.39329946	33.777366, -84.397387
33.774914, -84.399713	33.775895, -84.397149

Mobile Location Accuracy Continued

- The error is very high
- Though the geolocation module is easy to use, it is shown to be inaccurate and unreliable, especially indoors
- Alternatives are certainly available and need to be incorporated

Server Testing



- Server response time seems to increase linearly with the number of users
- The tests were performed assuming each user was in a single group
- Response time was averaged over twenty requests
- Requests were not concurrent

Lessons Learned

Most Interesting

- Calculating Centroid for large clusters
- Spatial Alarms and Notifications
- Cloud-Based Server

Overcoming Difficulties/Pain Points

- Finding out most optimal centroid calculation
  - Originally, we were overthinking how to calculate the centroid, creating complicated math equations. We eventually realized this and scaled back the complexity of the problem and found that just using the average of all the points produced the best results and most realistic “safe zone.”
- Creating a server
  - Debated whether to send requests from the server every x number of seconds and have the application reply with its coordinates or to have the application send a request every x number of seconds and replying with centroid, which may desync the updates of the apps. Determined the latter was the best way to go.

Connection to Class

- Spatial Alarms / Localization
  - Used concepts of localization with use of GPS to find location of an individual
  - Applied Spatial Alarms techniques
    - Chose circle spatial alarm to restrict difficulty of calculating if in boundary
    - Uses Euclidean Distance to calculate within spatial alarm
- Privacy / Security
  - Uses idea of friend based trust for sharing location
  - Add security measures to prevent unauthorized users from viewing locations
- Networking
  - Built cloud server to host the application

References

References

[1] https://www.cc.gatech.edu/~lingliu/papers/2007/geogrid-icdcs07.pdf
[2] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6256251
[3] https://www.cc.gatech.edu/~lingliu/papers/2008/Anand-SEDE08.pdf
[4] http://www.dgp.toronto.edu/~dearman/papers/2005-Rendezvous-CHI.pdf
[5] https://matplotlib.org/contents.html