Near

Ryan Batterman*
Department of Computer Science
California Institute of Technology
batterman@caltech.edu

Joseph Choi†
Department of Computer Science
California Institute of Technology
jichoi@caltech.edu

ABSTRACT
Near is a web-based mobile application for sharing posts bound with the location at which the post was made, and browsing posts through the use of a map.

Categories and Subject Descriptors
D.3.3 [**Language Constructs and Features**]: Frameworks; E.2 [**Data Storage Representations**]: Object representation; J.7 [**Computers in Other Systems**]: Consumer products

General Terms
Mobile application, Geohash

Keywords
Near, Android app, posts, Google Maps API, geographic coordinates

1. MOTIVATION
The authors of the Near application realized that they did not have a good way of figuring out what people around the world were saying at any point in time. Using the 'follow' feature of Twitter, one can see what individual users are communicating; but one cannot easily see what, for instance, people in the Egypt are saying about the Egyptian revolution. Near was created to solve this problem. Near lets users easily see what people in different parts of the globe are saying.

2. SCENARIOS
1. Ralph is a Junior at MIT who is taking a very tough, badly taught Physics class. He wants to vent his frustration anonymously, but Facebook does not let him do so – it requires that he have his name attached to all his posts. He could employ Twitter; but then none of his classmates would see his post. So Ralph decides to employ Near. Using Near, he is able to anonymously write about his frustration and share it with the entire world (not just his friends). Ralph writes his commentary, and is happy.

2. Robert is a Caltech student who wants to comment on David Politzer's antics with the rest of his Caltech Physics class. He could write it on Facebook, but Facebook only allows him to communicate the idea with his friends; he wants to share it with the people in his immediate vicinity. So Robert creates a post and puts it on Near. Other people in the class who are messing around on their phones at the time see his nearby post.

3. Alice is a high school senior who was admitted to Caltech and MIT. She already toured both of the schools, but she wants to get a better sense of how people at the two different schools compare. She opens up Near and sees MIT students complaining about their problem sets; by contrast, the Caltech students all write happy posts about their experiences on campus. She chooses Caltech over MIT.

4. Alex is touring Paris and wants to see what people have to say about the different tourist locations. So he opens up Near and sees what others post about the different monuments, museums, and restaurants. Near gives him upclose commentary about all the touristy parts of Paris. In fact, in Near, he sees a picture of a particular exhibit in the Louvre that greatly interests him. He decides to spend his day at that museum.

5. Helga is a reporter for the New York Times and wants to see what Egyptians are saying about the revolution occurring in their country. Twitter does not provide her an easy way to search by location; she can look in an ad-hoc manner for people who say, for instance, that they are Egyptian – but this does not allow her to see exactly where these people are in the country. It does not permit her to easily compare what people in Cairo are saying to what people in Suez are saying. In fact, it is quite hard for her to do any analysis whatsoever. So she opens up the Near application and is easily able to see what normal Egyptians are saying about the situation in their country. She analyzes this and submits her article to the New York Times.

6. Eva tried out a new coffee shop on Lake Street. It took her 25 minutes to get her coffee, and when she finally obtained it, it tasted terrible. Eva wants to rant about the shop so that no one else makes the mistake of spending money there. She could write a Yelp review. But she is lazy and wants instant gratification, so she opens up Near and quickly writes up her rant.

7. Eliezer is procrastinating on his PS 12 assignment at Caltech and wants some way to entertain himself while he is waiting. So he opens up Near and starts reading the funny, interesting posts to pass the time. After

*Ryan Batterman: Main front-end developer.
†Joseph Choi: Main back-end developer.

awhile, he decides to contribute his own comments – so he writes ironic posts about the class which are later read by other Caltech students.

3. EVOLUTION OF IDEAS

3.1 Initial Idea

Near was first envisioned as a single-page (single-tab) application. This page contained a standard map created using the Google Maps API. When the user selected a 'center point' by clicking on the map, posts around that point would appear on the screen. The bottom part of the application would contain a text box which, when selected, would expand and allow a user to type his post. A nearby 'send' button and the 'return' key would allow him to submit his post. Upon submission, his post would be displayed on the map next to his location, for the world to see.

We considered only allowing users to only see posts from their nearby area. We believed this would provide some degree of privacy to the application (e.g. Caltech students would only want their posts to be seen by other Caltech students) and we initially thought that people would not find much utility in reading posts from locations far away. However, we came up with (seemingly) plausible use cases to demonstrate that the second assumption was false. Furthermore, it was noted that this would create a major problem with network effects. If only Caltech students are using the application, and their posts are not displayed to MIT students, why would the first MIT student start using the application? He would have no content to see, so he would not use Near. Hence, we decided to allow users to see all posts.

3.2 Posts Tab

Soon thereafter it was realized that having an additional page containing a list of posts would be useful. A list of posts would allow the user to more easily read the posts from a particular location: instead of having to awkwardly zoom in and out to see all the posts in a particular place, the user would just have to select a location on the map by clicking and then scroll through the 'posts' page to see the local content. Thus we decided to add a page containing all the posts.

The above ideas coalesce into a two-page application which looks something like Figures 1 and 2.

We further realized that having the 'posts' page would allow us to order the posts based upon relevance to the user; instead of just giving the user all the posts around some particular location, we have some complicated function determining which posts the user will see; having this function gives us flexibility with respect to what the user is shown.

We noted that it would be easy to add upvote and downvote functionality in the posts page. This would allow us to use the feedback from different users to determine the relevancy of posts for the user. And it also gives users an incentive to put content into the application. Instead of just hoping that someone will read their posts and find them interesting and useful, users will get explicit feedback as to how well others liked their posts – in the same way that the 'like' feature of

Figure 1: Preliminary mockup of the Near application. The red marker specifies the center point.

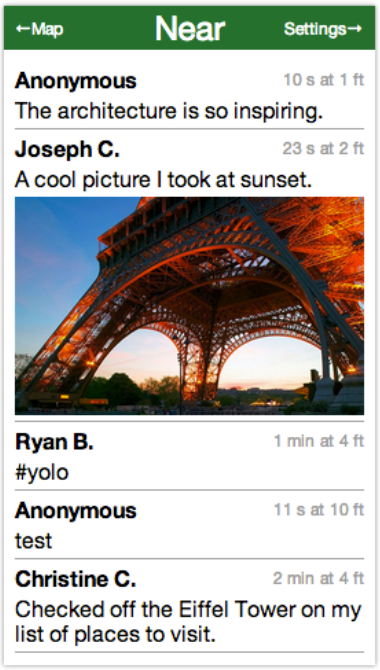

Figure 2: This was the early design for the posts interface.

Facebook and an innumerable number of other applications encourages users to continue posting, this feedback will encourage our users to put more content on Near. To display the upvotes and downvotes that a user received, we decided to add an additional page to the application for the posts which the user had submitted.

We also considered adding a display which showed the number of times that a particular post was viewed by other users. However, it was decided that this did not provide additional benefit over that provided by the upvote/downvote feature.

To further incentivize users, we considered adding 'follower' functionality similar to what exists on Twitter and Quora; if a user follows a poster, then he would start seeing the messages of the poster in his feeds. It was hypothesized that this would provide a way for users to filter for more relevant posts. But this feature adds a large degree of complexity to the application (we would need another feed for displaying posts from the people the user is following). Furthermore, if users want to follow other users who provide useful content, they can use Twitter; Near's geographical focus provides no additional benefit over Twitter.

3.3 Posts Ordering

We considered different forms for the ordering scheme. The ordering function would map the posts to a real valued 'relevance' number; posts would be displayed in order of decreasing relevance. The ordering function was imagined to take as input the distance between the selected map location and the post, the time since the post was created, and the number of upvotes and downvotes; closer, more recent posts with many upvotes but few downvotes would be more relevant. It was noted that the way in which the ordering functions weighted these things relative to each other would determine the character of Near. If the function highly weighted the upvote/downvotes of posts, then Near would start resembling Quora. Everyone would only see the highest rated posts. If, alternatively, the function put a high weight on the newness of the posts, then the application would become very transient and would begin to resemble 4chan. We further realized that the smaller the weight put on distance, then the less our application is differentiated from our competitors (like Twitter) – hence, we decided to put a high weight on the distance factor.

3.4 Selection Using Circles

We also considered adding circles to Near. The idea was that, instead of selecting a single center point (where posts which were closer to the center point would tend to have higher relevancies), the user could instead draw a circle on the map – posts which were in this circle would be displayed instead of just posts around the center point.

This would allow us to distinguish between users who wanted to search for posts within an immediate vicinity (e.g. someone who wanted to see what people are saying about the Eiffel Tower) and users who wanted to identify posts in large geographical areas (e.g. someone who wanted to see the top content of the entire United States); users would be permitted to give their degree of specificity. Thus we would be able to cater to a wide array of user desires.

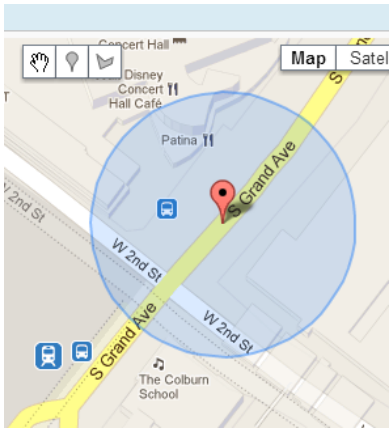

Figure 3: Circle on a Google Map. We considered using this feature in the Near application to allow users to give locational specificity.

There are drawbacks to this approach, however. Instead of just selecting a center point, the user is forced to place two markers (one to specify the center point and one to specify the radius). This additional complexity increases the amount of time that it takes the user to find posts, and it makes the application much more difficult for beginning users: they need to figure out how to draw circles and what these circles mean. We could possibly mitigate this issue by adding a tutorial to the application, but that makes the application harder to use. It is unclear if these costs outweigh the benefits of allowing users to do queries which have a low degree of locational specificity; after all, these sorts of queries do not distinguish Near from competitors such as Reddit (i.e. Reddit shows the top content from across the world; these low-specificity Near queries show the top content from large geographical areas – hence we are effectively competing with Reddit). Thus, we put the circle issues as a low-priority item and did not implement it in the first iteration of Near.

3.5 Search Bar

We also noticed that users would find it awkward to scroll through the map to find a particular location (e.g. the Eiffel Tower). Hence we decided to put a search bar into the application. We also decided that it would be useful to have swiping between the pages of the application – so we added that, as well. And we created a login page to allow users to become associated with a particular identity. The application at this point is shown in Figures 4, 5, and 6.

4. SERVER-SIDE

4.1 Web Framework

The back-end was coded with Node.js. Node.js is a software platform where applications are developed in JavaScript, which is run internally by the Google V8 engine. The V8 engine compiles JavaScript into native machine code before executing it. From the outside, the use of JavaScript makes it appear that the code is interpreted, as no manual compilation step is necessary. This is done behind the scenes by Node.js. This also prevents changes in the code from being visible without restarting the application, so it doesn't pro-

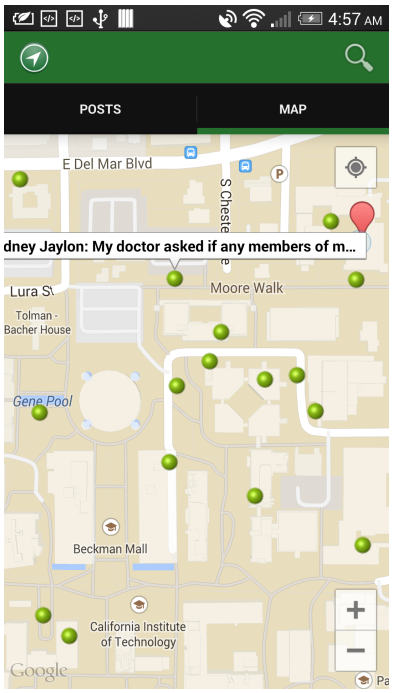

Figure 4: Map interface in the Near application. The green dots correspond to posts; the red marker is the center location; the search icon expands to a search bar.

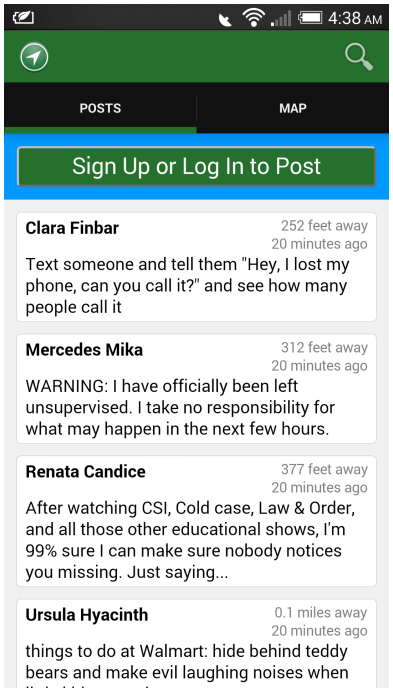

Figure 5: Posts feed page. This page displays the posts of nearby users, and upon login, allows users to post. In the future, the upvote/downvote feature will be implemented here.

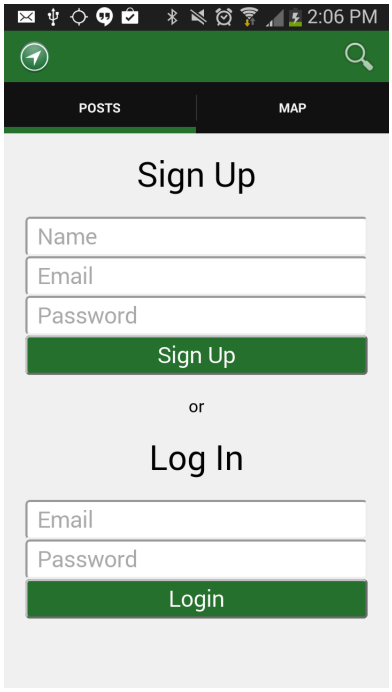

Figure 6: The signup/login page of the Near application.

vide this advantage of interpreted languages. However, because the code is actually compiled to machine code, rather than being converted to bytecode or being interpreted directly, this allows the server to be fast.

In Near, we used the asynchronous event features of Node.js to maximize throughput and efficiency. Accesses to the database, reading files, and computing proximities from Redis are all asynchronous, using an event system to call a callback function on finishing. This means that other computations could keep running while waiting for the data to be retrieved from the database. This allows one server to be able to handle many concurrent users at the same time.

4.2 npm

Node.js provides the benefit of the back-end code being written in JavaScript, and a significant portion of the front-end code is also written in JavaScript. This allows a substantial amount of code re-use between the front-end and the back-end, which accelerates development. Traditionally, the front-end and the back-end would be written in separate languages; popular server-side choices are Python, Java, and PHP. Furthermore, the vast amount of libraries in Node.js managed by an easy-to-use package system called npm, the Node Package Manager, allows easily downloading of relevant libraries, and a dependency tracking system, which alleviates these complications. There is no need to re-invent the wheel with the vast collection of libraries for Node.js.

Node.js provides a rich set of libraries that greatly expedite web development. There are simple wrappers for creating and starting a server, which allows one to quickly develop server-side code and iterate on it. Express 4.0.0, a web application framework for Node.js, was used to handle the logic

of resolving URLs, serving static files, and providing error handlers.

4.3 Data Store

For storing the data, Redis was used. Redis is an in-memory data store that maps keys to several different kinds of values, including strings, lists, sets, and sorted sets. Redis is written in C, and stores the whole dataset in memory. This allows reads and writes from the data store to be fast. With Redis, we used an AOF system. This system puts any changes made in the data store to an append-only file, such that writing changes to disk was fast and corruption-resistant. The AOF was set to sync every second, which means that if the server unexpectedly goes down due to an power outage, then at most only 1 second worth of data is lost. Furthermore, the data that is changed is only appended to the end of the file, which means that if the server went down while data was being appended to the file, only the last append would be corrupted. This could be fixed such that Redis discards this corrupted entry, and thus we have a working backup with less than 1 second worth data lost.

4.4 WebView

The posts user interface is used to show users the posts closest to their selected location on the map. Users can view the posts that are made without having to be signed in. However, in order to post, users must either create a new account or sign in if they already have an account. In order to do this, the user clicks on the "Sign Up or Log In to Post" button at the top of the posts user interface. After signing in, the user may now post, and the user's current location will automatically be associated with the post. The background is light blue, which was chosen to be the same color as the marker indicating the user's current location on the maps fragment. This decision was made in order to indicate that the user is posting at his or her current location.

The user interface for posts is done by rendering a webpage in Android using WebView. Using a WebView allows for rapid development, using familiar tools such as HTML and CSS in order to markup content and design the user interface. Furthermore, it takes a while for updates to the Android app to get on the Google Play Store. This inhibits updates to the way a page is display, changes in the way the data is sent to the client, and changes in the posts functionality. However, by placing this in a WebView, these updates can be done independently of the process for uploading apps for download on the Google Play Store.

Using a WebView allows the use of JavaScript to program the user interface. Ajax (Asynchronous JavaScript and XML) is used to asynchronously fetch data from the server. Retrieving posts is done by sending an asynchronous request with the location of the marker to the server. The server then fetches the posts data from the data store, sorts them by distance, and returns the posts data. The JavaScript in the WebView is then used to update the posts user interface with the new posts.

When fetching posts, the new posts have to be added to the map. This is done by transferring the geographical coordinates of the posts to Android native code through the

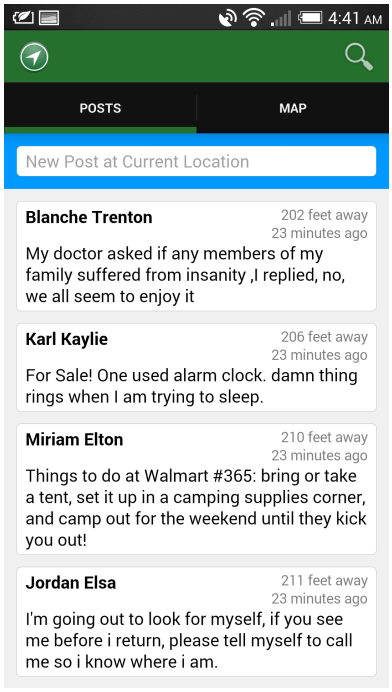

Figure 8: View after signing in or logging in.

use of a `JavaScriptInterface`. There is a special class in the Android code that responds to specially marked code in the webpage's JavaScript, which then are forwarded to the Android code. This method is used to transfer the coordinates to Android, which then marks the coordinates of the retrieved posts on the maps fragment.

4.5 Fetching Algorithms

Consider the problem of storing geographical coordinates $c = (latitude, longitude)$ inside a database represented by a set of geographical coordinates C, such that one can efficiently apply the fetching operation: get the set of stored geographical coordinates $C' \in C$ within a given distance d of a geographical coordinate c.

The naive approach would be to store the geographical coordinate c in an array representing C, and given a geographical coordinate c, we loop through the array, computing a distance from each each coordinate in C. We have

$$C' = \{c' \in C | dist(c, c') \leq d\}$$

A different algorithm involves generating a quad-tree mapped to the sphere. The globe is separated into four sections, which are recursively divided into four sections such that each section has at most one geographical coordinate. This tree structure could then be traversed in order to find all the posts within a given distance of a geographical coordinate in $O(\log |C|)$ time. However, this approach was not taken.

4.6 Distance Formula

Note that the distance operator cannot be naively defined as $d'(c, c') = \sqrt{\Delta latitude^2 + \Delta longitude^2}$ since latitude and longitude are isomorphic to $\phi \in [0, \pi]$, $\theta \in [0, 2\pi]$, and the

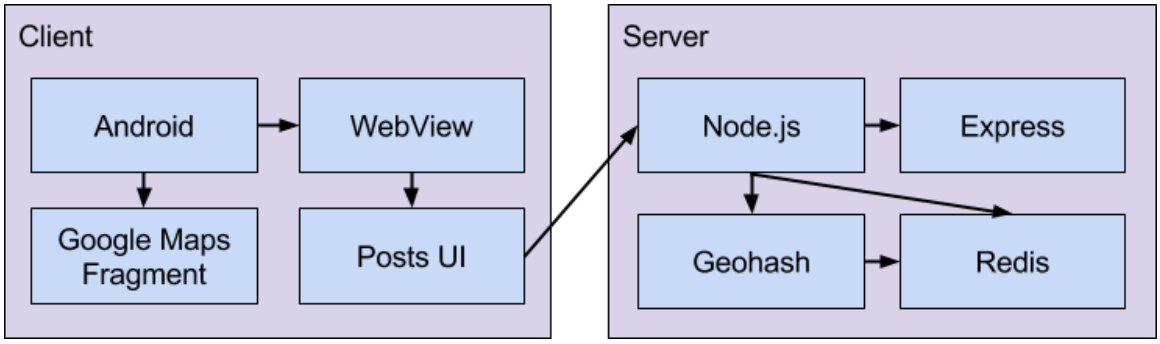

Figure 7: Diagram of the overall front-end and back-end architecture.

area element $d\Omega = \sin\phi \, \mathrm{d}\theta \, \mathrm{d}\phi$ is a function of ϕ. The correct distance formula between coordinates $c = (a, b)$ and $c' = (a', b')$ is

$$\frac{d'(c,c')}{R} = \begin{cases} 2\arctan\left(\sqrt{\frac{z}{1-z}}\right) & 1-z>0 \\ 2\arctan\left(\sqrt{\frac{z}{1-z}}\right)+2\pi & z\geq 0, 1-z<0 \\ 2\arctan\left(\sqrt{\frac{z}{1-z}}\right)-2\pi & z<0, 1-z<0 \\ \pi & z>0, 1-z=0 \\ -\pi & z<0, 1-z=0 \end{cases}$$

where $z = \sin^2\left(\frac{a-a'}{2}\right) + \cos a \cos a' \sin^2\left(\frac{b-b'}{2}\right)$

and $R =$ radius of the Earth

Note that the naive algorithm of computing distances and finding all the geographical coordinates with $d'(c, c') \leq d$ takes $O(|C|)$ time where n is the number of geographical coordinates in the database. This does not scale well with millions of posts, with many users.

4.7 Geohash Algorithm

A special Geohash algorithm was used in order to implement putting 2-dimensional geographical coordinates in a 1-dimensional database, with efficient fetching of nearby geographical coordinates. The algorithm supports the following operations

- Insert a $(latitude, longitude)$ coordinate into the database.
- Fetch all $(latitude, longitude)$ coordinates within a given $(latitude, longitude)$ in $O(|C'|)$ time, which is generally much better than $O(|C|)$ time.

The algorithm works as follows. In order to insert a geographical coordinate $c = (a, b)$ into the database, the geohash h of even length n for c must be computed. For the latitude a, construct the binary array $a_0a_1a_2...a_{n/2-1}$ of length $n/2$ as follows: let $l_0 = -90°, r_0 = 90°, m_0 = (l_0 + r_0)/2$. If $a < m_i$, then set $a_i = 0$, $r_{i+1} = m_i$, otherwise set $a_i = 1$, $l_{i+1} = m_i$. Repeat for the longitude b, except set the initial variables $l_0 = -180°, r_0 = 180°$. The geohash is then computed by interspersing the binary arrays for a and b: $h = a_0b_0a_1b_1a_2b_2..a_{n/2-1}b_{n/2-1}$.

This actual geohash is used as a key for a post entry in the data store. For the case of Near, this entry is stored in a sorted set, where the keys are sorted by their value. We thus have a linear data store where posts are ordered by their respective geohashes. The geographical coordinates within a distance d of a geographical coordinate c can then be found by computing a hash h of c, and then looking at the first few binary digits of h, $h_k = a_0b_0a_1b_1a_2b_2..a_kb_k$. k is chosen such that the next geohash h'_k of h_k, where $a'_k = a_k + 1$ which is carried to increasing i until $a_i = 0$, and similarly for b'_k, represents a distance from h_k of at least d. All the geographical coordinates whose prefixes are the same are within this block. Then the distance formula for geographical coordinates can be used to find the set C'. For edge cases where c is close to the edge of a block, the 8 neighboring blocks are computed and retrieved, and then the distance formula for these geographical coordinates can be used to find the set C'. This algorithm runs in time $O(|C'|)$.

5. RELATED PRODUCTS

Secret is a mobile application for iPhone and Android that lets you share sentiments and pictures anonymously with your friends. It recently got $10 million in funding at a $50 million valuation. The difference between Near and Secret is that Near sends messages to people nearby, and these people are not necessarily your friends. Furthermore, Near is more of a real-time application that lets you find out what's going on right now, at this moment, at the place you're at. Thus, Near is a location-based anonymous sharing application, while Secret is a friends-based anonymous sharing application.

Whisper is a mobile application for iPhone and Android that lets you share sentiments and pictures anonymously with the entire world. It recently got $30 million in funding at a $200 million valuation. The difference between Near and Whisper is that Whisper is not a real-time stream of incoming messages, rather, it is a series of messages in boxes that were sent. Near shows you the posts that are the most recent as well as the closest. Whisper has separate sections for nearby posts (which are not the most recent), and the latest posts (which are not nearby). Thus, Near is more of a real-time and location-based sharing application, while Whisper is more of a generic application for sharing things anonymously with the world.

6. FUTURE WORK

There are several features which could be implemented to further improve the Near application.

At the moment, users can specify only a single center at a time. This forces them, if they want to stream posts from multiple locations, to switch back and forth between the different positions – a nuisance for the user. We could fix this by allowing the user to save pins. This may add additional complexity (and hence make it more difficult for new users to understand the application), though.

It may be useful to add messaging to the application. If Alice sees that Joe consistently posts high quality content to the application, then she may want to communicate with him individually. We left this feature out of the initial product, because of lack of time, and because we thought this might make Near to similar to our competitors. But it may, nonetheless, be worth implementing.

It was noted that the color scheme of Near does not work well for users with color blindness; these users tend to have difficulty distinguishing between red and green, and hence are probably unable to differentiate between the red center marker on the maps page and the posts. If so, this makes using the application difficult to use for the approximate 10% of color blind people. It may, in fact, be more than 10%, because computer users are color blind at higher rates than average members of the population.

We could integrate picture functionality with the Near application. Pictures allow users to express certain things that they cannot through simple text – for instance, if someone wants to display a particular building on Caltech, doing so with text is impossible. This feature could probably be implemented with little effort, little downside, and lots of upside.

We could use machine learning techniques to improve the relevance of the posts displayed to users. One option would be to feed the upvote/downvote values into a collaborative filtering algorithm to try to predict whether a user will like a post or not. An issue with this approach, however, is the sparsity of the data; most of the posts will have few upvotes – so predicting by, for instance, a straightforward application of the support vector decomposition algorithm on the posts probably would not work well. We could also try natural language processing techniques; if e.g. a user liked all 100 posts which referenced 'food', then we could determine that a new post with the word 'food' in it will probably be liked by that user. Naive Bayes would be the first classifier to try, because it is quite simple and has a reputation of working quite well on text classification tasks.

To get more people using the application, we could try asking more friends to use the application. Once we have a few useful or funny posts on Near, new users will have better experiences on the application and hence attracting more people to Near will be easier. After that, growth may occur organically (i.e. the current users bring new people to the application), or it may not. If it does not, then we could try altering the search terms in the Google Play Application description to target specific keywords. At the moment, we are not targeting any particular keywords – so users are not finding Near through the 'search' feature of the Play Store; fixing this may bring many new users. Alternatively, we could try setting up advertisements for Near. This could require a fair amount of capital and hence VC involvement; hence we would employ this option only as a last resort.

7. CONCLUSION

If Near takes off and becomes widely used, then people will easily be able to share their sentiments, thoughts, and pictures with all the people near them. People will be able to use Near to share relevant information about the situation with the people near them, start conversations, and possibly even make new friends.

8. ACKNOWLEDGMENTS

The authors would like to thank Adam Wierman for his valuable input on various aspects of the project, including the design of the user interface and the structure of the backend code.