Damage Tracker - A Cloud and Mobile System for Collecting Damage Information after Natural Disasters

Chris Hodapp, Matt Robbins, and Jeff Gray
Dept. of Computer Science
The University of Alabama
Box 870290
Tuscaloosa, AL 35487-0290
(205) 348-2847
clhodapp@crimson.ua.edu
mrrobbins@crimson.ua.edu
gray@cs.ua.edu

Andrew Graettinger
Dept. of Civil, Construction,
and Environmental Engineering
The University of Alabama
Box 870205
Tuscaloosa, AL 35487-0205
(205) 348-1707
andrewg@eng.ua.edu

ABSTRACT

Tornadoes and other natural disasters frequently cause large amounts of damage to buildings and infrastructure. An important part of learning from these events is assessing key damage-indicators within the affected area. Researchers can analyze these damage-indicators to better understand the event and how to minimize future loss. These assessments require many teams of researchers, government agencies, and volunteer citizen groups to survey affected areas and collect information. Assessment teams take thousands of digital photographs for later review. When combined with GPS data, these images can be used to document and understand an extreme event. In this paper, we present *Damage Tracker*, a software system for capturing and managing tornado damage information. A mobile application for Android devices allows users to capture, annotate, and upload geo-tagged photos. A web application stores uploaded photos and meta-data and displays this information on an interactive map. This system allows an online community of users to easily share data, which encourages more collaboration, reduces duplicate collection efforts, and potentially improves the quality and depth of subsequent research. *Damage Tracker* realizes the benefits of crowdsourcing and citizen science in the context of disaster data collection.

Categories and Subject Descriptors

J [**2**]: Engineering

General Terms

Management, Design

Keywords

tornado, natural disaster, cloud, geo-tagging

1. INTRODUCTION

Assessing damage after natural disasters is an important part of response and recovery and also provides an opportunity to learn from an event. In the short term, this information can help first-responders and disaster management agencies determine which areas were hit hardest and decide how to distribute resources (e.g., medical relief and temporary shelter). In the long term, although it is impossible to prevent natural disasters, researchers can learn how to better design infrastructure by collecting and analyzing data about past disasters. This knowledge can provide valuable insight to governments, city planners, and engineers. The ability to predict the behavior of disasters can enable governments to be better prepared, which may minimize human casualties. Civil engineers study the destructive forces of natural disasters, such as earthquakes and tornadoes, to recommend stricter building codes to ensure that buildings can withstand intense ground shaking or high wind speeds.

In this paper, we present our experiences developing *Damage Tracker*. We describe the major open-source technologies and tools used to build this application and give a brief reasoning for each choice. We conclude with some future goals for this project, including generalizing this work for any type of natural disaster.

2. BACKGROUND AND MOTIVATION

Damage assessments are done using qualitative measurements made from visual observations. The Enhanced Fujita (EF) Scale for Tornado Damage is a two-step system developed by the National Oceanic and Atmospheric Administration (NOAA) [1]. The first step is identifying the type of property being observed (e.g., “one- or two-family residences”, “Strip mall”, ”softwood tree”, etc.) The system refers to this as the *Damage Indicator* (DI). Each DI has an associated ordered set of specifiers called *Degrees of Damage* (DoD), which are numbered in order of increasing intensity. Each unique pair of DI and DoD represents the localized strength of a tornado, which is expressed in wind speed (mph). The expected wind speed determines the EF rating of a tornado. Although a range of wind speeds and EF ratings are associated with a tornado, the largest sustained estimated wind speed is used to assign a signle EF rating for a tornado. Ratings can be determined by an expert in the field or from a photo captured by any researcher [4].

It is necessary to associate these ratings with accurate geo-location data when constructing spatial and temporal models. Today, many mobile phones and digital cameras have GPS receivers installed. These receivers allow the device to measure a user's location with good accuracy (about 10 meters) and encode this information in the picture. This location data is embedded in the meta-data of image formats, such as JPEG (EXIF).

On April 27, 2011 an EF-4 (Enhanced Fujita scale) tornado [5] devastated Tuscaloosa, Alabama, killing 43 people [3] and causing total devastation to businesses and residential properties in 12% of the city. In one study of the Tuscaloosa tornado, researchers showed how systematically mining crowd-sourced information could be used to help first responders. A public API from the popular photo sharing site Flickr was used to search for geo-localated images tagged with the keywords "Tuscaloosa" and "tornado". The search results were then used to present damage along the path of the tornado [2].

Damage Tracker is a new software application developed at The University of Alabama to address the lack of solutions for easily capturing and sharing tornado damage information. The April 25-28, 2011 tornado outbreak, which affected many areas in the Southern and Midwest United States, served as a principle motivator for this project. In total, 358 tornadoes were confirmed by the National Weather Service in 21 states, making the April 2011 tornadoes the largest outbreak on record. The April 2011 disaster raised two problems with the way damage information is collection. First, assessment teams do not use standard equipment or methods for storing photos and GPS data to enable sharing. In one case, proprietary software tools have been developed to combine the data from a GPS receiver with images from the camera by comparing time stamps. Second, there are no public web sites for uploading and sharing this data to better coordinate surveying effort. The equipment required and lack of special-purpose sites for sharing also limits the contributions from the public. *Damage Tracker* uses commodity hardware and software to build a system for collecting and sharing this data. A mobile application, built for the widely available Android platform, has been developed to capture photos and annotate them with damage indicators, degree of damage ratings, and descriptive notes. A web application accepts uploaded reports from the mobile app and displays them on a custom Google Map. The web application can also accept photos uploaded through the browser and parses their GPS data.

3. DATA MODEL FOR DAMAGE TRACKER

Researchers are interested in estimating and assigning each tornado an EF rating. The wind speed of a tornado is typically derived from the resulting category and degree of damage in an area. It is therefore critical that photos of these affected areas be captured immediately after an event before any debris has been cleared or reconstruction efforts are initiated. Areas in the path of a tornado may be unrecognizable, so it is very important that an accurate location be associated with each image.

The data model is central to the design and function of *Damage Tracker*. To give the user more flexibility, in addition to images and GPS data, users may also add notes to an image and assign them EF Damage Indicators and Degrees of Damage using the mobile app, prior to image upload. The specific information that *Damage Tracker* collects for each snapshot includes the following: 1) Image, 2) Latitude and longitude tuple, 3) NOAA Damage Indicator, 4) NOAA Degree of Damage, 5) Street address, and 6) Notes.

The mobile application stores this meta-data in a SQLite database. When images are uploaded to the website, this data is transfered to a MySQL database. Images are stored external to the databases within a photo directory. The web application uses a MySQL relational database to store persistent meta-data. Image locations are displayed on an interactive map by markers. One of the requirements of this application is that it must manage data from multiple natural disasters, separated temporally and geographically. In order to manage this for the user, we introduced the concept of "Collections", which are user-defined labels, dividing images into unique sets.

4. DESIGN AND IMPLEMENTATION OF DAMAGE TRACKER

This section summarizes several of the implementation and design issues that were investigated during the development of *Damage Tracker*.

4.1 Web application

A web application was developed for *Damage Tracker* using the Play Framework[1]. Play is a web framework that supports applications written in the Java and Scala languages. Play with Scala was used because each has a strong focus on improving developer productivity. We were also strongly influenced by the wide variety of open-source libraries available within Java's large ecosystem.

4.1.1 User-Visible Pages

Our web application allows the user to interact with several pages:

Map Displays Google map populated EF-labeled markers for each photo (see Figure 1)

Photo Uploader Allows users to upload new image files, which appear in the photo queue (see Figure 2)

Photo Queue Allows users to preview photos before adding them to collections

Photo Metadata Editor Allows the user to view and change metadata attached to photos

4.1.2 Architecture and Models

Play is a Model-View-Controller (MVC) web framework, following the design popularized by Ruby on Rails. This means that the application is segmented into three components:

Model The *Model* component is responsible for interfacing with the application's various data backends (most often, a MySQL database). This layer consists mostly of our application's Object Relational Mapping (ORM) framework, which is defined using Play's

[1]http://www.playframework.org

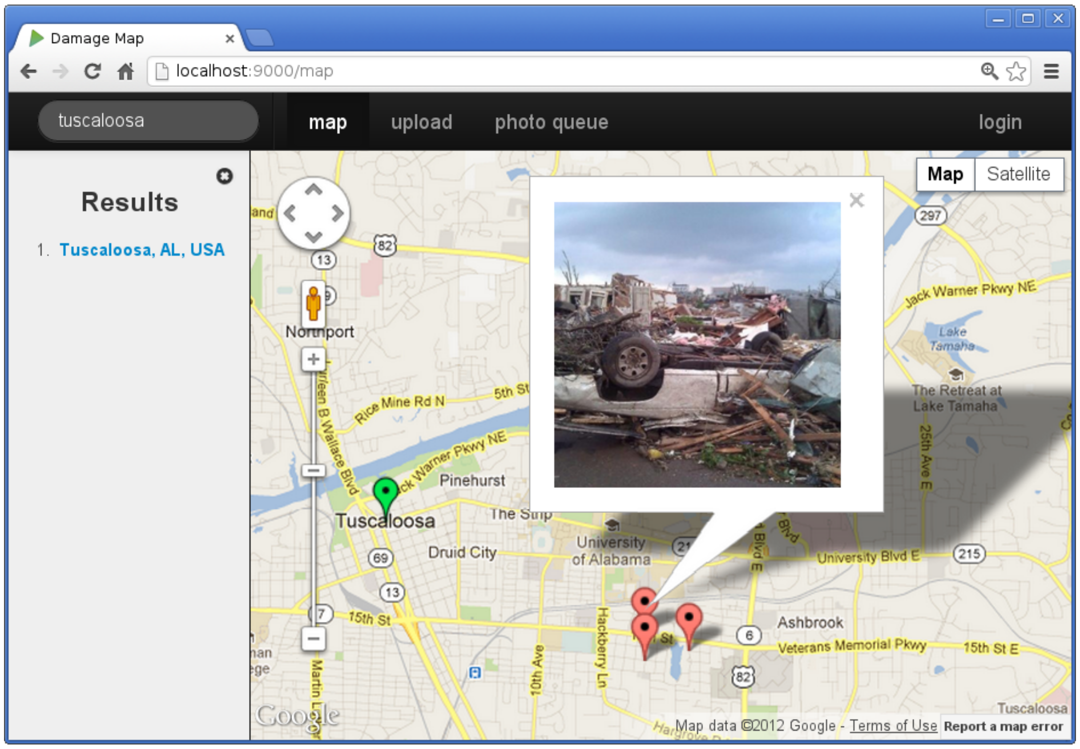


Figure 1: Web application map screen

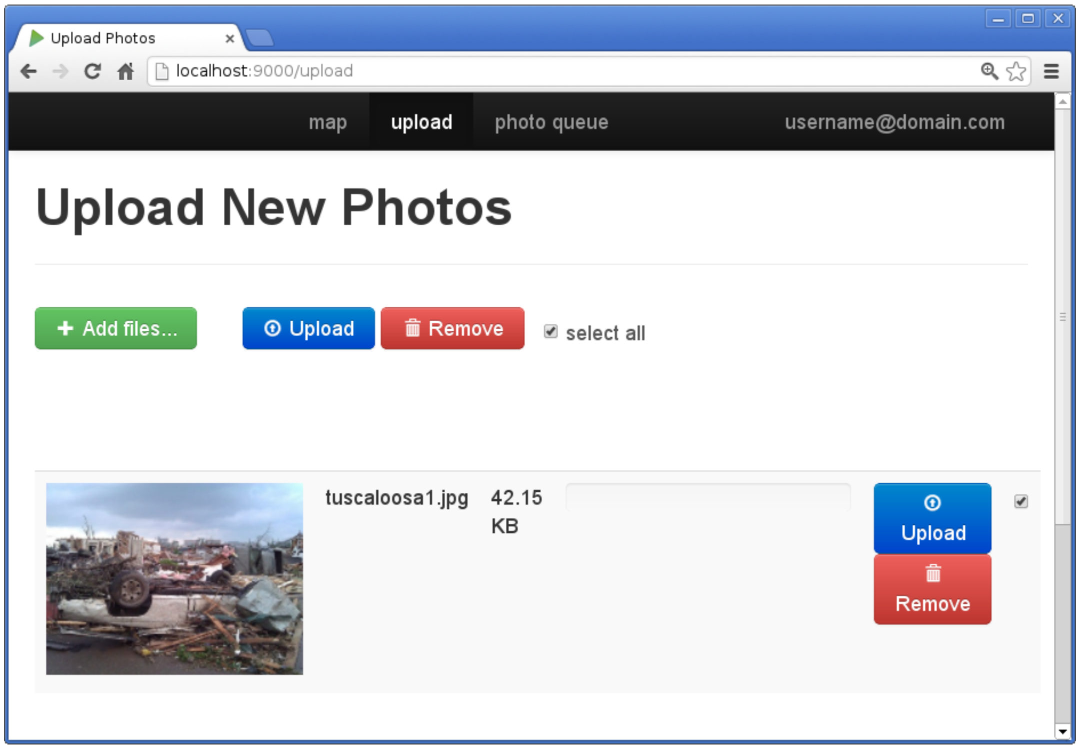


Figure 2: Web application upload images screen

Anorm database access library. The *Model* is also responsible for providing a bulk storage API to the rest of the application. Currently, bulk storage backends are implemented for local (hard disk) storage and cloud (Amazon S3) storage.

View The *View* component consists entirely of web page templates, defined using Play's built-in template framework. A web page template is a file containing interleaved web markup (HTML) and application code (Scala). This allowed us to define dynamic pages, which build themselves using information encoded in traditional Java objects.

Controller The *Controller* component is responsible for accepting page requests from the web server and returning the correct *View* for that request. For example, for a request for "map.html", the *Controller* component should return the contents of "maps.scala.html". Frequently, the *Controller* must interface with the *Model* in order to fulfill a request. This is because a *View* often requires data from the model in order to populate itself.

4.1.3 Play Framework

Play offers a number of features that reduce the amount of time spent recompiling code and watching error consoles. One such feature is "hot reload". This means that whenever a page is requested, Play checks whether the source code for that page has been changed. If it has, it is recompiled before the page is loaded (on the client side, this simply looks like a slow page load). Play also offers in-browser error messages. This means that if there is some error in the compilation or execution of the code behind a page, an error line number or a stack trace is provided attractively in the developer's web browser. These two features combined to ensure that we could spend almost all of our development time either editing code or testing that code in a web browser.

Play's bundled libraries and Scala's flexible syntax combine to make asynchronous (multithreaded) programming extremely easy. Play offers a data type, called a Promise, which encapsulates computation that will return a value of a given type (or error). Application code can easily transform Promises with functional programming techniques (Promise is a Monad) or attach callbacks to be run when the encapsulated computation completes (or immediately if it's already completed). The key feature of Promises, though, is how easily they can be created (and thus, how easily the developer can begin using multiple application threads). In order fo faciliate creating Promises, Play provides a function called Future, which accepts a block. The user need only pass a block encapsulating the computation that they want to run asynchronously to Future and it will return a Promise of the block's return value. The system will run the block's code on a worker thread at its next convenience (with the return value fulfilling the promise). This API allowed us to make all of the blocking operations in our web application (disk IO, database access, etc.) run on worker threads, which allows our application to handle significantly more concurrent clients.

4.2 Cloud Deployment and AWS

The original deployment strategy for *Damage Tracker* was a single server model. The application was hosted on a dedicated Windows or Linux server and ran a local instance of the MySQL database. Furthermore, all uploaded images were stored on local volumes (e.g., high capacity hard disk drives). The reasoning behind this strategy was that there would be separate instances of the Damage Tracker service, deployed on a different server, for each event, thereby limiting the demand on any given instance. This approach seemed feasible with only a small group of users and an upper bound on the number of photos (e.g., a few thousand). However, natural disasters, especially tornadoes, can happen quickly and with little warning, so there would not be preparation time to manually deploy new instances of the app and advertise availability before first responders would need access to the system.

Another major focus of this project was to create an improved repository for data used in disaster research. The depth and scope of this research is largely dependent on the sampling of sufficiently large data sets that are statistically significant. Separate instances would restrict the view of disasters to localized events and users would have to access multiple services to analyze relationships. This presented two engineering challenges when designing the web application. First, we had to consider how the app would manage tens of thousands of user images in a way in which storage space could scale dynamically and that also protected against data loss from disk errors or destruction. Second, the web service runtime would need to scale with the traffic of hundreds of users and properly handle concurrency issues.

In order to address these challenges expediently, we looked to cloud computing to provide us with the required flexibility and capbilities. Specifically, we considered three products in the Amazon Web Services (AWS) ecosystem: EC2 for hosting the application process, RDS for hosting a relational database, and S3 for file storage.

4.2.1 Amazon EC2

The web service is hosted on an Amazon Elastic Compute Cloud (EC2) instance. EC2 is a cloud service that provides remote computing resources using virtualized, multi-tenant servers that can be accessed from anywhere on the Internet (e.g., a web server with a public IP address listening on TCP port 80). One of the major benefits of using EC2 is easy scalability of computing power. In a single-server system, a host process can handle a finite number of requests per time until it is overwhelmed by traffic and either becomes unusably slow or unreachable. Unlike a single-server system, when an increase in traffic to an EC2 instance might impact performance, an administrator can clone a new image and AWS will automatically use load balancers to distribute the work between available nodes. This means that during tornado season in the US, when tornado activity peaks and many events overlap, the site will maintain the same quality of service and reliability as if only one event had occured.

All of the AWS products we used were simple to provision. For development, we chose to deploy on the free tier of EC2 which provides a micro instance with up to two Elastic Computer Units for short processing bursts, for a maximum uptime of 750 hours per month. Amazon offers a selection of virtual machine images to install on an instance. We selected an Ubuntu 12.04.1 LTS 64-bit distribution because it was the closest match to our development environment.

4.2.2 Amazon RDS

The web service requires a MySQL database for structured (relational) data storage. In our current deployment, we take advantage of the free tier of Amazon Relational Database Service (RDS) to fill this role. Like EC2, RDS offers several tiers of service (ranked by size qualifiers, such as "small", "medium", and "large"), but only a single instance in the micro tier is available for free. The free tier is also limited to hosting databases no larger than 20GB in size.

In addition to offering this free service tier, RDS has several other key features. First, RDS offers the customer the ability to change the service tier ("size") of a database instance on demand. This would allow an administrator to rescale a deployment to current demand with great ease. Second, RDS automatically creates periodic database backups (a single backup, updated daily for the free tier), which significantly reduces the likelihood of data loss due to a single system's (inevitable) failure. Our RDS deployment was nearly turnkey; Amazon RDS MySQL databases are addressed over TCP using the native MySQL protocol at a URL (read from the AWS management console). Because Play supports database configuration in the main application configuration file (*application.conf*), using an RDS-hosted database rather than a local one was a simple matter of changing three configuration lines (the server URL, the connection username, and the connection password). Since Play applications can apply database schema updates automatically, *Damage Tracker* was able to create its tables on its first startup.

4.2.3 Amazon S3

The prototype for *Damage Tracker* was designed for a single server and image storage and was implemented using the local file system of the host. However, this created several limitations for fail over and recovery. Most concerning was the single point of failure for irreplaceable research data if the physical disk became corrupt or damaged. Hosting the web app on EC2 meant the user data would need to be just close to the compute node in EC2 as it was to the local host in a single server. Data is likely to be stored in an Amazon data center so it should be more easily replicated and geographically distributed than local data. To provide local storage to our prototype EC2 node, we mounted an Amazon Elastic Block Storage (EBS) partition to the virtual machine, which acts like physically-attached storage. EBS is a low-level storage-as-a-service. The root partition of each EC2 machine resides in a small, free EBS volume. Additional EBS volumes can also be purchased and mounted to provide EC2 applications with bulk storage.

This model, however, had some severe limitations. The data had to be backed up manually in snapshots, because there was no automatic replication or distribution across data centers. Fortunately, AWS provides another storage-as-a-service called Simple Storage Service (S3) which was significantly better-suited for our application. In trying out S3, we wanted to retain the flexibility to return to local volume storage in the future, so we used an interface to abstract over storage backend differences in our code. Specifically, we defined an interface called StorageBackend with two concrete implementations: LocalStorageBackend for single-server volume-backed hosting and S3StorageBackend for interfacing with Amazon's Simple Storage Service (S3). The S3 backend was implemented using RhinoFly's S3 module[2], which provides a simple API to S3, allowing us to integrate S3 functionality into our application. S3 uses a flat hierarchy of "buckets" similar to directories (but not recursive) for storing files by a key or filename. One challenge in using S3 was all users share the same bucket namespace. That is, all bucket identifiers must be globally unique. Presently, we overcome this issue by asking the user to define a unique prefix for their buckets in the application configuration file. The StorageBackend interface provides three functions to the application for storage:

- store(bucket: String, file: java.io.File, key: String)
- lookup(bucket: String, key: String)
- delete(bucket: String, key: String)

4.3 History of Damage Tracker

4.3.1 Pre-Existing work

In a previous semester, a senior design team focused their project on using mobile apps to help document tornadoes. The team of computer science students included Kyle Redding, Luke Taylor, and Jonathan Fikes. They built an Android smartphone app for collecting data damage information in the field. The app allowed the user to take photos of damage scenes. After capture, an edit activity appeared and the user was prompted to annotate the photo with typed notes or address information. The app stored images on the device's internal SD card and the meta-data was stored in a SQLite database. However, there was no way to retrieve this data from the app or push it to a remote host. We extended their app with this functionality and also added NOAA indicators to the annotation options.

4.3.2 Modifications Leading to Current Version

Images are uploaded using the account credentials that were created on the web site. The mobile client stores the server URL and user email from the most recent upload, but does not store the user's password, as this would be a security risk. Images are uploaded via HTTP POST requests, which are sent to the web service. Each image is uploaded in a seperate POST in order to minimize the chance that an upload failure for a single image interferes with the upload of any other image.

We also designed a streamlined, modal interface for the application, giving the user the choice between the app's three main functions at startup (Figure 3) and providing additional important information to the user on the information entry screen (Figure 4).

5. CONCLUSION AND FUTURE WORK

Damage Tracker was originally intended as a system for studying the EF strength of tornadoes by using the domain-specific NOAA rating scale. This resulted in assumptions made to the database schema and application design that does not make this system fit for studying other natural disasters. We believe, however, that with small modifications to the data model, the mobile app and web service could be generalized for mapping other disasters and location-specific events.

[2]https://github.com/Rhinofly/play-libraries

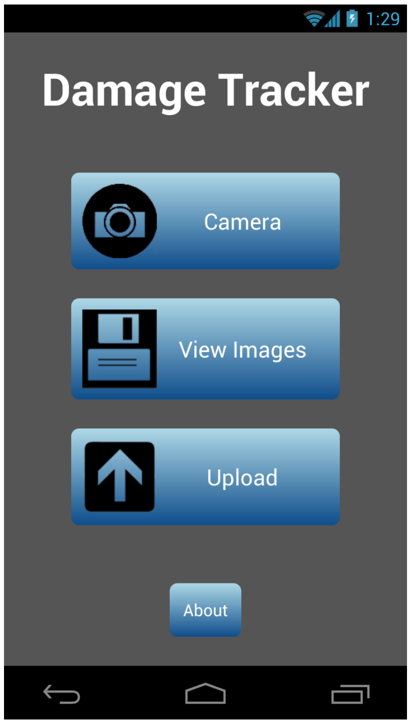
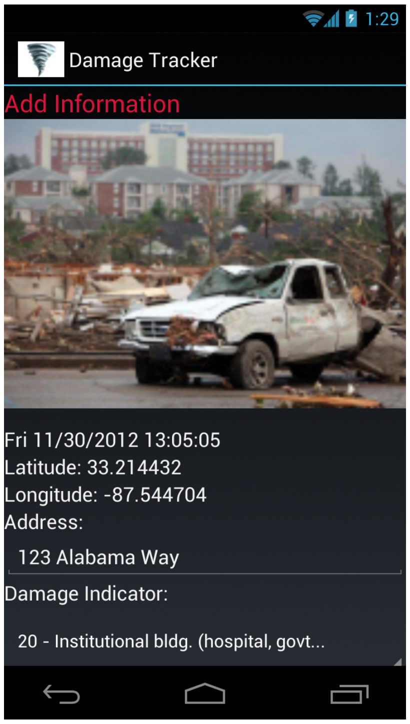


Figure 3: Android application home screen



Fri 11/30/2012 13:05:05
Latitude: 33.214432
Longitude: -87.544704
Address:
123 Alabama Way
Damage Indicator:
20 - Institutional bldg. (hospital, govt...

Figure 4: Android edit information screen

The current state of the system described in this paper has several limitations that future work should address. As previously discussed, a generalized model for annotating geo-tagged images and events should be realized and implemented to expand the usefulness of this app. Also, the decision to use the Android platform for creating the mobile uploader was made to build on prior work and our familiarity with the SDK. By developing versions of the client for other mobile platforms (e.g., iOS and Windows Phone), the community of potential contributors could grow substantially. Developing a version optimized for a mobile browser is another possibility.

6. ACKNOWLEDGEMENTS

This research is supported by NSF RAPID award #1047780.

We would like to thank Kyle Redding, Luke Taylor, and Jonathan Fikes for their early vision and work on the prototype for *Damage Tracker*.

7. REFERENCES

[1] R. Edwards, A. Dean, R. Thompson, and B. Smith. Convective modes for significant severe thunderstorms in the contiguous united states. part iii: Tropical cyclone tornadoes. *Journal of Weather and Forecasting*, 27:1507–1519, December 2012.

[2] R. Fontugne, K. Cho, Y. Won, and K. Fukuda. Disasters seen through flickr cameras. In *Proceedings of the Special Workshop on Internet and Disasters*, SWID '11, pages 5:1–5:10, New York, NY, USA, 2011. ACM.

[3] J. Morton. Tuscaloosa county death toll from tornado increases to 43. *Tuscaloosa News*, 1 June 2011.

[4] S. Potter. Fine-tuning fujita. *Weatherwise*, 60(2):64, 2007.

[5] D. Prevatt, J. van de Lindt, A. Graettinger, W. Coulbourne, R. Gupta, S. Pei, S. Hensen, , and D. Grau. Damage study and future direction for structural design following the tuscaloosa tornado of 2011. Technical report, Center for Advanced Public Safety, 2011.