UNIVERSITY OF CALIFORNIA

Santa Barbara

Aether

Real-time Recovery and Visualization of Deleted Tweets

A Project submitted in partial satisfaction of the requirements

for the degree of Master of Science in Media Arts & Technologies

by

Pehr Lorand Hovey

Committee in charge:

Professor George Legrady, Chair

Professor Lisa Jevbratt

Professor Alan Liu

June 2010

The project of Pehr Lorand Hovey is approved.

George Legrady, Committee Chair

Lisa Jevbratt

Alan Liu

June 2010

Aether

Real-time Recovery and Visualization of Deleted Tweets

ACKNOWLEDGEMENTS

Working with a topic that spans multiple disciplines can be a daunting task. My project committee was instrumental in helping me approach the topic from multiple angles. Professor Legrady continually pushed me to improve the visual component; which greatly increased the clarity and impact. Professor Jevbratt helped me consider protocols as art, and how the behind the scenes data systems are just as important as the visual product. Professor Liu provided a digital humanist perspective on the data analysis and supported me on the RoSE project, which gave me the programming experience necessary to make this all happen.

I owe thanks to Larry Zins for providing network support to get my system online and serving data to the world. Allison Schifani and Dana Solomon helped start all this Twitter stuff when we worked together on the Urban Sensorium project in Professor Liu's Literature Plus class. Karl Yerkes helped me consider how the users would feel about machines breaking social contracts, an interest we share. Ali Van Dam helped proofread and put up with the never-ending stream of deleted tweets on my screen.

Finally, I am grateful to my parents for sewing the seed of graduate education by sending me to undergraduate and encouraging me to explore my passions.

ABSTRACT

Aether

Real-time Recovery and Visualization of Deleted Tweets

by

Pehr Lorand Hovey

Aether is an exploration of the irrevocability of speech in online social networks. Just seconds after posting something online it has likely been disseminated to dozens of people and definitely been archived by an unknowable number of automated systems. Though a *delete* button may provide solace to those having second thoughts, in reality it is a façade—you can never truly take something back online. And yet, people around the world are constantly changing and removing things they have said, altering their online image.

Aether investigates this phenomenon of online self-erasure by capturing and visualizing deleted Twitter updates in real-time. Whereas people might hope that their deletions go unnoticed, *Aether* amplifies and dissects the act for the public to see.

Aether is comprised of a central data processing server and multiple client applications that interact with the server using Open Sound Control (OSC). The Ruby server continuously processes the Twitter Stream API in a multi-stage pipeline. The system stores all tweets in a local Memcached instance. Deletion notices in the API stream are crosschecked with the archive to recover the data that has been deleted from the public API. Deleted statuses are pushed to visualization clients in real-time. Clients can be developed on any platform that can communicate using OSC. The visualizations prepared for this project were written in Java using the Processing framework. In practice the system is capable of recovering several deleted tweets per minute, within seconds of the user deleting it.

TABLE OF CONTENTS

LIST OF FIGURES

1 Introduction

Aether is an exploration of the irrevocability of speech in online social networks. Just seconds after posting something online it has likely been disseminated to dozens of people and definitely been archived by an unknowable number of automated systems. Though a *delete* button may provide solace to those having second thoughts, in reality it is a façade—you can never truly take something back online. And yet, people around the world are constantly changing and removing things they have said, altering their online image.

Aether investigates this phenomenon of online self-erasure by capturing and visualizing deleted Twitter updates in real-time. Whereas people might hope that their deletions go unnoticed, *Aether* amplifies and dissects the act for the public to see. Though the original tweet was publicly available, deleting it may seem more like a private act expressed between the user and social network. When unseen systems capitalize on this event it questions our understanding of public and private space online and how much control we really have over our online identity.

Twitter presents a new and interesting opportunity for multimedia artists to explore communication at a global scale. Since starting in 2006, this "micro-blogging" service has exploded in size. Twitter users send 50 million *tweets* (status updates) per day, many of which also include precise geolocation information. Though status updates are limited to 140 characters, they come with a wealth of other metadata ready to be examined and visualized by today's digital artists and researchers. These updates are often personal in nature and can be a useful gateway into examining the human condition in the 21st century.

Another benefit for multimedia artists is the low barrier to entry for participating in Twitter. The service is free and provides a standard web interface. Mobile tweets can be sent using standard mobile text messaging (SMS) as well as rich-client applications on internet-enabled mobile platforms like iPhone, Blackberry and Android. Using Twitter in an installation piece means that the artist can elicit audience participation without requiring the development of one-off mobile applications that may be platform dependent. Instead users can simply send a tweet using the software they may already be familiar with.

1.1. Problem Statement

Data permanence on the Internet is a core topic explored by *Aether*. We are prolific creators of social content that has often has a very timely component---talking about what we are doing, expressing our current emotions. As new events and postings are published the old ones fade away into history. We do not often think about the fact our data is likely retained forever by the social networks we have interacted with. Unlike physically written words that we might be able to definitively destroy, it is effectively impossible to truly erase something that has diffused into the cloud of the Internet.

Surveillance is another theme touched on by Aether. Users do not realize that they are being observed by a remote system such as this one. While millions of people allow their Twitter updates to be public, they likely do not expect anyone beyond their friends to be actively viewing them. They consider their social network of followers as a semi-private space inside the otherwise very public Twitter ecosystem. The boundary of public and private space online, explored by artists such as Lisa Jevbratt [1], is ripe for discussion. Twitter estimates that there are over 100,000 third party applications accessing their API, which begs the question: would people say the same things if they knew they were being automatically watched and archived by entities other than their followers?

Twitter is considered by many to facilitate peer-to-peer conversation beyond the original idea of one-way microblogging and status updates. With this interaction paradigm comes concepts borrowed from traditional human conversation. People may be concerned with their public image and carefully groom their profile to ensure it projects the right message. In Twitter, as in real conversation, you can’t truly take something back after you have said it—but it doesn’t stop people from trying.

1.2. Relevance of the Research
Current academic research on Twitter and social networks in general primarily focuses on how users interact with others and what types of things they say. Not much has been done with the question of why people delete things after they have been published. *Aether* takes on this question while engaging other social aspects that have been mapped to the online space such as issues of privacy expectations.

On the technical side, *Aether* represents research into methods for high-volume real-time data acquisition and processing. It tests various methods for implementing a multi-stage pipeline and dealing with a never-ending data stream. The client-server setup of separating the data gathering components from the end product provides an example for other similar systems that need to have independent components situated on multiple networks.

2. Related Work
This project operates in both the technical and social realms and has influences and parallels in multiple fields. Though Twitter is the medium for data acquisition, the core principal of *Aether* is examining people and their actions online. While there has been much research focused on online publishing, not much has been said about online self-erasure and the implications of people regretting what they said online.

2.1. Human Condition Online
The explosion of online publishing and social interaction has produced works that look at a broad spectrum of emotions online. *We Feel Fine* [1] acts as a barometer of people expressing feelings online. It regularly scrapes thousands of blogs from around the world and looks for people expressing their feelings. The resulting data is presented in six separate visual forms that provide detailed breakdowns on the types of emotions being expressed. There is also an API that allows others to create works using their data.

Deleting a tweet can be considered an implicit expression of regret. *Secret Regrets* [2] posts user-submitted regrets in a blog format. *PostSecret* [3] publishes pictures of user-submitted secrets, usually in the form of anonymous postcards sent through the mail. While the content and tone varies widely, many of the published cards consist of an admission of something the sender regrets in some fashion.

REGRETS [5] by Mulfinger and Budgett is a multi-year project that asks people to anonymously submit short descriptions of things that they regret. The project initially solicited input from physical booths and mobile computing platforms that went directly to people. The website continues to accept new input and serves as an ever-growing archive of regret around the world. *REGRETS* approaches this subject by focusing on specific, quantifiable regret as vocalized by the feeler. *Aether* takes a different angle by looking at implicit examples of regret (people going back and erasing something they said), and leaves it to the viewer to decide what the person was regretting. Unlike *REGRETS*, the subjects do not realize their actions are being recorded which may avoid response-bias at the cost of less precision since we can only infer what their thinking may be.

2.2. *Research on Twitter*

Twitter has attracted many researchers in the humanities and social sciences eager to discover new insights into how people interact online. Much of this research has focused on why people choose to use Twitter, and how they interact with other users of the system.

Why we twitter: understanding microblogging usage and communities [5] is a paper written just one year after Twitter started that uses social network analysis to estimate user intention. Huberman et al. took social network analysis further by looking at the follower vs. friend ratio as a predictor of user behavior [6]. The paper particularly examined how certain subsets of users' social networks matter more than their network as a whole when it comes to posting behavior. Other research has investigated conversation and collaboration on Twitter, including the use of the '@'

symbol to denote a screen name [7]. They reveal how people use Twitter for conversations and to what extent the current design of Twitter may hamper true collaboration.

2.3. Saving Online Data

Aether can be considered an archive of Twitter data with a real-time reactionary component. Archiving online material is as old as the Internet itself. Internet search engines function by crawling the web and storing copies of webpages in vast databases to be retrieved later. Though search engines like Google have recently begun to provide some ‘real-time’ results, most search engine data is hours if not days old.

Many search engines allow users to view a cached copy of each page, usually corresponding to the most recent version the engine has seen. Search engines tacitly acknowledge that their data is likely ‘stale’ and that the page may have changed or been removed since it was indexed. There is no declaration of which pages are in fact deleted from the public view, but the pages will usually disappear from the search engine index over time. Oftentimes items that have been recently deleted or hidden can still be viewed by browsing a cached version of the page if it is still in the search index.

Unlike most search engines, the Internet Archive [9] provides multiple historical snapshots for most websites, allowing users to observe how a website looked at specific points in time. Their goal is to create complete snapshots of the web and their data is often not available online until months after it has been acquired.

Twitter provides their own search service, making old tweet data available to the public. This service has a historical limit of roughly two weeks and they specifically remove deleted tweets from their index, though they almost certainly maintain a permanent copy internally. The United States Library of Congress has also recently announced [9] that they are archiving every tweet, including historical data going back to Twitter's nascence in 2006. While they will likely provide interface tools for the data, it remains to be seen if they will make deleted tweets available.

Tweleted [10] was a service that exploited a former bug in Twitter's systems to find deleted tweets for a specific user, entered on the website. Sometimes deleted tweets were not removed from the search API so they would turn up in a comparison with the regular lookup API. Twitter has since fixed the issue and the service no longer works.

In some sense, we are all temporary archivers of online data since our web browsers store temporary copies of every visited webpage in a cache on disk. For some users, cache clearing is infrequent and they can build up a large collection of outdated webpages.

2.4. Software for Data Aggregation & Visualization

Aether does not intend to dictate a fully new visualization framework or data processing paradigm but the development of the visualizations was inspired by other work in the field. Several software systems have been created to facilitate generalized data processing and visualization, especially in the area of aggregation of large datasets.

Yahoo! Pipes [11] is a novel web interface that uses a visual programming environment to facilitate data manipulation. Users can connect many operations and data sources together to get their desired composite dataset. This data can then be exported in various formats including JSON and XML. It is especially useful for people not comfortable with more traditional programming. Unlike *Aether*, the data becomes available as a complete set after the pipe is run, not as a continuous stream.

The *Manyeyes* portal [12] from IBM is a collection of online data aggregation and analysis tools that allow anyone to upload a dataset (or use some existing data) and explore it with many different visualization tools.

Behaviorism is a cross-platform C++ framework by Angus Forbes [13] that provides several services to support real-time information visualization. It uses a scene graph, data graph and timing graph to give the developer control over every aspect of a complex dynamic visualization piece.

Prefuse [1] is a Java library that provides tools for creating interactive data visualizations. It provides several data structures for storing and searching the data as well as several different kinds of data plots and filters to transform the data and the visual presentation. Prefuse Flare is a recent Flash implementation of the same concept.

3. Recovering Deleted Tweets*Twitter API Overview*

Twitter is one of many web services that provide a free *Application Programming Interface* (API). An API is a set of specific methods or commands that allow other developers access to their data. The Twitter API allows authenticated third parties to retrieve tweet and user data as well as send new tweets. Twitter provides three (currently separate) APIs that each serve a distinct purpose. Each API makes data available in JSON (JavaScript Object Notation), XML and sometimes RSS Atom format, depending on the particular method.

The Twitter Search API allows you to search for tweets containing specific keywords. Additional parameters allow you to specify a geographic region (expressed as a radius around a latitude & longitude point), a date range and the number of results to return.

The Twitter REST API is the main interface for enquiring about specific users and tweets. REST stands for REpresentational State Transfer and is a design paradigm that specifies how to structure the methods of a web-service API. This interface has methods for retrieving all data for a specific user as well as their most recent tweets and the tweets on their timeline (what they would see in their personal Twitter client). Applications can also send and delete tweets on behalf of a user, provided they have the proper password.

The Twitter Streaming API is the newest and most groundbreaking interface. It takes advantage of long-lived HTTP connections to send real-time data for a potentially unlimited amount of time. Once connected, tweets come streaming at you until you manually disconnect. The full Twitter stream (dubbed the Firehose) is only available to large-scale “partners” such as Google or the Library of Congress but the sampled stream still provides an avalanche of data. The publicly available stream is said to contain a uniform 1/20th sample of all tweets [15]. This interface provides parameters to filter by keyword or by a specific group of users. This API is the most difficult to use efficiently since it involves maintaining a potentially unreliable connection and processing data in real-time as it arrives. If data is not processed fast enough, Twitter reserves the right to terminate the connection.

Aether uses the Streaming API to get real-time tweet data and deletion notices. The REST API is used to look up additional information about a user.

3.2. Metrics for Understanding Deleted Tweets

Twitter provides a wealth of metadata with each tweet that is a useful starting point for investigation of the deletion event. Additional metrics can be derived from the provided data to allow for even more in-depth analysis. Some metrics are useful for quantitative evaluations while others are more useful for conveying general information about the subjects in question.

Below is a summary of the principal items examined by *Aether*.

Context: A tweet is not necessarily an island --- it might be part of a larger conversation. When the deleted item is placed into conversational context it can sometimes be quite obvious why it was deleted (such as in the case of a typo, with a corrected version re-sent immediately). Whereas a single tweet may not seem that interesting, viewing more elements of a persons conversation at once can provide a fascinating window into their lives. When a tweet is recovered, *Aether* retrieves a few tweets sent by this person before and after the deleted one from the Twitter REST API in order to provide context.

Hashtags & Mentions: *Hashtags* are folksonomic markers used to self-categorize tweets based on content. This allows anyone with similar interests to easily find the tweet alongside all others with the same tag. Most hashtag usage is ephemeral and subject to ambiguity (the same tag meaning multiple things) or fragmentation (multiple different tags for the same thing) but some research has looked into using them for more powerful semantic applications [18].

Technically, a hashtag is any text without spaces, preceded by a "#" symbol. They are often used to tag a tweet about a specific timely event, such as The Oscars (#oscars) or a particular concert (#okgomay222010). The presence of one or more hashtags in a tweet would suggest that this person was engaging in a global conversation beyond their close circle of followers. In effect, they amplified their voice prior to deleting what they said.

Mentions are a way of tagging a specific user by prefixing their screen name with the '@' symbol. On most twitter clients there is a column or view that automatically displays other people's tweets that mention you. Mentioning users is primarily how people engage in a conversation directly with one or more people. Deleting a tweet with a mention in it is the equivalent of trying to take back something that was said face-to-face in real life, though there is no guarantee that the other person saw the mention.

Followers_count & friends_count: The number of followers is useful since it is the number of people that will immediately receive any tweet sent, prior to it being deleted (though no guarantee they will see it before it is deleted). Follower counts are also interesting because the number is out of direct control of the user—other people have to voluntarily follow them and can always un-follow them, driving the number down. Friends are the people this user is following. Some researchers have looked beyond raw counts to see how the ratio of followers to friends may be a predictor for user behavior on Twitter [8] and spam detection [19].

Lifetime: The length of time that a tweet was available for the world to see before it was deleted is an interesting metric because it alludes to the (unknowable) amount of people that may have seen it before it was rescinded. Longer lifetimes suggest that more compelling reasons than fixing a simple typo that could have been seen immediately.

Location: Twitter users can enter their physical location to be displayed on their profile, though there is no requirement for accuracy or even that it be a real place. Unlike Twitter's tweet geotagging features, this location typically refers to a hometown or other regional designation. The location is geocoded to precise latitude and longitude using Google Maps geocoder service when possible.

Source: The specific software used to post the message to Twitter can be used to estimate if they were at a desktop computer or using a mobile device.

Time sent, time deleted: The creation time is recorded from the initial tweet data (reported in UTC time). The deletion time is recorded when the deletion notice is received, also in UTC time. Times can be converted to user's local time zone by using the UTC offset.

Typo text: The Levenshtein edit distance formula[18] is used to compare the tweet to the set of other tweets that were sent by this user before and after this one. If the distance is low (<5) for any of the tests then it is likely the deleted tweet was considered a typo. The tweet that replaced this one is retained for analysis.

User description: The self-reported "about me" section is useful to humanize them and get a sense of their writing style.

UTC Offset: The time difference (in seconds) between the user's local time and UTC / Greenwich Mean Time is used to estimate the user's current time zone.

Word Frequencies: Counts of how many times every word has been seen in all deleted tweets are useful to see if a tweet is similar in content to other tweets that have been deleted.

Additional data beyond what is described above is retained to round out the visualizations and facilitate experimentation with the data. A detailed rundown of all data is in the Appendix.

3.3. Designing a Data Gathering Framework

The data-gathering portion of *Aether* consists of a long-running software system written in the Ruby [17] programming language. Ruby is known for its "natural" programming style and the brevity of its code. Ruby also strives for general cross-platform compatibility and can run on most modern operating systems. There are many implementations of the Ruby language including one that runs on the Java Virtual Machine (JRuby) and one by Microsoft that can run in a web browser (IronRuby). The “standard” implementation is called Matz’s Ruby Interpreter (MRI). The current stable version of MRI is 1.9.1 while the 1.8 branch is still widely used and supported. *Aether* was developed on MRI 1.9.1.

The *Aether* server has the joint duty of continuously gathering and storing new data while also serving data to multiple visualization clients, a structure that requires concurrent programming. Data is shuffled through multiple steps in a modular pipeline that all execute in parallel. Each stage runs as a completely separate Ruby process and communicates using Distributed Ruby (DRb) [19], a core Ruby component that allows any Ruby object to be shared over the network. In theory each stage could run on a separate computer, though for simplicity they are all run together on the same testing server.

The *Aether* pipeline consists of three primary stages: *Grab*, *Retrieve* and *React*. These stages operate in Producer/Consumer relationships where each stage produces data that is consumed by the next stage in line. Stages receive data via a *push* paradigm whereby the producing stage remotely populates the target stage’s input queue, rather than a *poll* method where the target stage would periodically check upstream for new data.

Unlike some Producer/Consumer systems, there is no way for a consumer to tell the initial producer, Twitter, to temporarily pause data production in the event that the consumer cannot keep up. Moreover, the volume of data coming into the system is not constant and may include temporary spikes that must be accounted for to avoid data loss. Thus it is imperative that each stage process data as fast as possible to avoid losing data. Each stage must process different volumes of data with the input volume generally decreasing in later stages as unneeded data is discarded. Figure 1 shows the data flow in the *Aether* server with sample volumes from a thirty minute test run at mid-day.

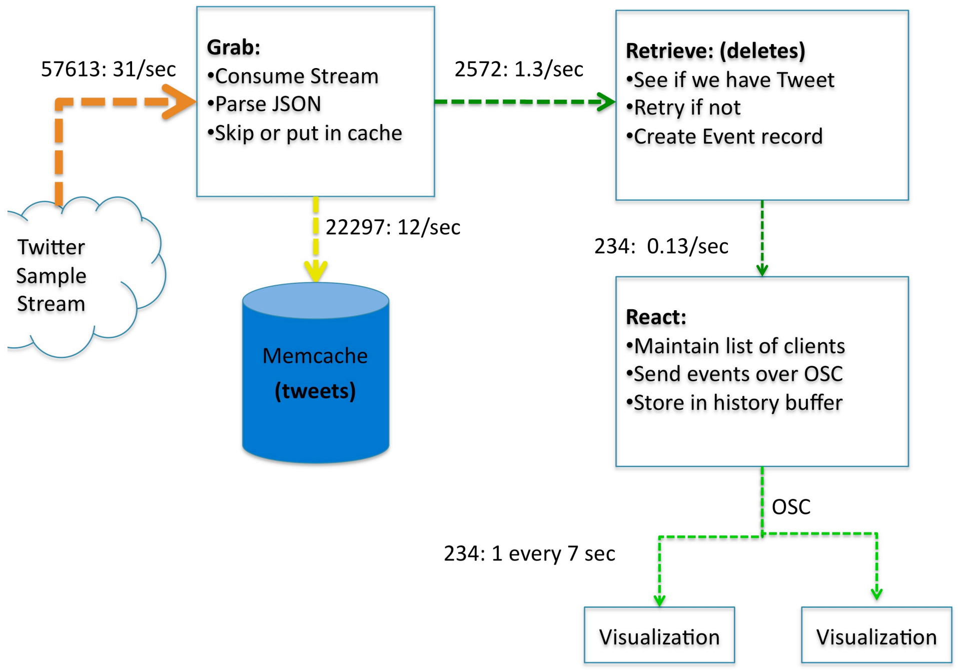


Figure 1: *Aether* server data flow

3.3.1. Grab: receiving and storing Tweets & Users

The *Grab* stage operates the connection to the Twitter streaming API. Each line of data initially arrives as a raw string of characters that must be converted into a usable data structure. *Aether* requests data in JSON format since it is less verbose than XML, saving bandwidth and memory resources. Twitter currently sends two types of data through the streaming API: status objects and deletion notices.

Deletion notices contain the id of the status that was deleted as well as the id of the sending user.

```
{:delete=>{:status=>{:user_id=>69114405, :id=>8408482002}}}
```

Figure 2: Sample Twitter deletion notice (JSON)

All delete notices are immediately forwarded to the *Retrieve* stage for further processing.

Each status object contains metadata for the status update as well as information regarding the sending user. An excerpt of a typical status object is in Figure 3.

```
{
"id": 13094530000,
"text": "Ok back not getting too bad. Hopefully I'll get some sleep.",
"created_at": "Thu Apr 29 22:53:35 +0000 2010",
    "source": "<a href=\"http://ubertwitter.com\"
rel=\"nofollow\">UberTwitter</a>",
    "user": {
      "friends_count": 127,
      "lang": "en",
      "created_at": "Thu Nov 20 02:36:19 +0000 2008",
      "statuses_count": 7038,
      "time_zone": "Edinburgh",
      "profile_link_color": "0000ff",
      "geo_enabled": true,
      "followers_count": 63,
      "location": "Anywhere you are!!",
      "screen_name": "fifinoir",
      "name": "fiona",
      "id": 17502304,
      "utc_offset": 0,
    }}
```

Figure 3: Excerpt from Twitter status object (JSON)

An important function of the *Grab* stage is to discard statuses we are not interested in to save resources. Subsequent stages in the pipeline may perform computationally expensive processes on each status so it is imperative to minimize the amount of work to be done and not lose desirable data due to buffer overflows. Each status object is run through a special skip function that determines if we should discard it or pass it on to later stages.

The criteria for skipping a status depend on the scope of the project and the intended uses of the resulting data. It is also important to minimize the complexity of the skip function since it will be executed more than any other processing function in the data pipeline. More specific and intensive tests can be deferred to later stages that process comparatively fewer data objects. The skip function has been shown in practice to discard as much as 60% of the data, saving considerable resources.

Automatically discarding data can be risky depending on the application requirements so it is important to choose the skip criteria carefully. Details on the assumptions underlying the skip function criteria are presented in the Design Assumptions section.

All statuses that pass the skip test are stored in the local memcache instance for later retrieval, keyed to the unique tweet id.

3.3.2. Retrieve: Looking Back into the Archive

Deletion notices are consumed by the *Retrieve* stage. The memcache is checked for the tweet id provided in the deletion notice. If successful, the tweet and accompanying User are retrieved for further processing. If the tweet was not found, the delete notice is put into a special queue to be tried again one or more times in the future. This is important because Twitter states that sometimes a delete notice may arrive before the status does, and there is also the chance that the status has been received but not saved yet.

This stage is able to perform computationally complex data processing operations since the maximum throughput is orders of magnitude lower than earlier stages. Any fields that will be made available to visualization clients are extracted and used to create a new record called an Event. Each Event corresponds to a successfully retrieved deleted tweet and is stored in the MySQL database so it can be easily retrieved and sent to external clients. Each event is also immediately sent to the next stage to be dispatched to currently active clients.

3.3.3. React: Interacting with Client Applications

All real-time communication with external clients is done using Open Sound Control (OSC) in the *React Stage*. This stage accepts connection requests and maintains a list of active clients that can each be sent specifically tailored data. The connection API details are covered below in the Client Applications section as well as the Appendix.

React maintains a list of the most recent several events to facilitate populating new clients when they come online. This list is initially pulled from the Event database on startup. The total size of the history list is capped so the oldest events are eventually discarded from the buffer.

Each fresh Event is popped from the incoming queue and immediately broadcast to all active clients. This enables each client to receive new deleted data in near real-time. The new Event is also added to the top of the history list to keep it up-to-date. *React* also immediately broadcasts server DataRate updates since they are time-sensitive.

3.3.4. Utility Stages

There are two utility stages that support the overall functionality of the system.

StageMonitor periodically requests a detailed status report from each main stage. The report describes the total number of items processed and approximate processing rates. This information is available to the server operator to see how each stage is doing. With each stage running as a separate process it would otherwise be difficult to get a summary of all stages.

RateMonitor calculates real-time data flow rates in each stage at a finer granularity than *StageMonitor.* Clients can use this information to visualize the state of the system. Rates are calculated as a rate per second over a change in time equal to the report period.

```
{:rate=>true, :now_ms=>1274754288000, :status_rate=>19.6141,
:del_rate=>4.230, :succ_rate=>0.05234}
```

Figure 4: Example of RateMonitor output

RateMonitor runs once per second to get an estimate of current data rates. The current timestamp is bundled with the rates in a Ruby Hash and forwarded to the *React* stage for transmission to clients.

3.4. Client applications

Client applications are any other software system that consumes deleted tweet data from the *Aether* server. Clients could include dynamic visualizations, re-broadcasters or even hardware devices with scrolling LCD screens. This project does not attempt to dictate exactly how a client application should function, but the components needed for successful interaction with the server API are documented.

The *Aether* server is designed to maintain connections with an arbitrary number of client applications that have registered to receive data. The server pushes new deleted tweets to all registered clients as they are recovered, enabling near-real-time visualization. The client-server API is built on Open Sound Control (OSC) since it provides for push-style data flow [20]. Pushing data to the clients as it is created is important since the flow rate is not consistent and it avoids wasting resources to constantly check the server for new information. The Server has a well-defined OSC address space that acts as the external API for the deleted tweet data. Clients must implement a similarly structured OSC address space to provide specific endpoints to receive new data. All OSC data is sent across the network using the User Datagram Protocol (UDP) since it is simple, well supported and efficient.

Clients must first register with the server to enable the server to send data back automatically. This is accomplished by sending the client's IP address and active port number to the server's */register* address. These details are used to assemble a unique identifying key, which is required for all subsequent server interactions. Multiple clients from the same IP address can operate simultaneously so long as the ports are unique. The visualization clients developed for this project random ports within a certain range to minimize the chance of collisions.

The IP address and port must be accessible by the server in order to establish successful bi-directional communication. This is not always possible, such as when the client is behind a router on a private network. In these situations a server-side technique known as UDP Hole Punching [16] can be used to trick the network into allowing the communication. On the client side, Universal Plug-n-Play can be used to automatically open the required port if the network hardware supports it. This method was successfully used in the Java-based client applications via *UPnPlib* from

SBBI [23]. These methods are not needed when both the client and server are on the same network but may be required if a client is being run offsite such as at a gallery.

Clients should send periodic */hello* keep-alive messages to inform the server that they are still running and expecting data. This is useful if the server restarted and no longer has a record of the active client. Messages from unknown clients will cause the client to be re-registered using the provided client key, re-establishing the connection. The time of the most recent keep-alive message is recorded for auditing purposes. This feature is designed to maintain reliable communication between the server and many clients when left unattended for long periods of time.

There are three main OSC methods for requesting event data from the server. */event/latest* will send the most recent event back, while */event/random* will return a random event from the history buffer. Since the history buffer is a fixed size and constantly updated, even the random event will be relatively recent. */event/history* will request the entire history buffer. This is useful to bootstrap a fresh visualization so it does not start empty.

Event objects from the server are transmitted as serialized JSON, as shown in Figure 5. The client must parse this into a usable representation. The event contains all the fields present in the server's Event object stored in the database.

Periodic updates from *RateMonitor* are sent as serialized JSON to the */rate* address of each client.

```
{"tweeted_at_utc":"2010-05-24 22:17:12 UTC",
"tweet_id":"14650753602",
"text":"@KristalMeth1 I'm sorry if you took any of my tweets personal..
They weren't directed towards you and that's my word.. I love you",
"screen_name":"I_Am_Spades","location_str":"Sandy Springs, GA, US",
"lat_lng_mapped":"0.372195987447699,0.698947592592593",
"sent_time_24":"Mon-17:17","deleted_time_24":"Mon-18:01",
"lifetime":2630,"source":"UberTwitter",
"friends_count":408,"followers_count":406,"user_deletes":1,
"word_text":"i m sorry if you took any of my tweets personal they werent
directed towards you and thats my word i love you",
"hashtags":[],"mentions":["@KristalMeth1"],"urls":[],"utc_offset":-
18000,
"description":"Praying for my enemies.. In Nomeni Patri Et Fili Spiritus
Sancti..",
"profile_image_url":"http://a1.twimg.com/profile_images/919651086/108060
789_normal.jpg",
"created_at":"2010-05-24 16:01:08 -0700","history":false}
```

Figure 5: Event data sent to clients (JSON)

4. Visualizing Deleted Tweets

While catching and storing deleted tweets is the primary mission of *Aether*, it is useful to see what you have caught. Two dynamic visualizations were created to help the public investigate the deleted tweet data.

A principle goal of visualizing information is to “make visible the invisible” [18]—to take a collection of data points and derive metrics and figures that reveal underlying features and patterns. The large and multidimensional nature of the dataset presented a challenge of scope and scale—what part of the data should be visualized? How can many tweets be visualized at once? The client-server design is intended to support many heterogeneous clients simultaneously, which reduces the pressure to make a single optimal visualization. Instead multiple visualizations can be presented side-by-side which increases coherence by promoting visual linking—an important concept when dealing with such multivariate data. This passage from *Graphics of Large Datasets* summarizes it well:

> Linking is only possible if multiple views of the same data can be displayed simultaneously [...] It is essential to think of many displays contributing to an overall picture and not to aim for some “optimal” single display [17].

Visual linking is promoted within a single visualization by designing everything to occupy one screen instead of switching between multiple disjointed pages. The visual elements themselves take the form of widgets—standalone objects that can be easily laid out on the grid and controlled independently. The goal is to make it quick and easy to re-arrange and resize components to explore different aesthetics. Each widget has the concept of a “current event” which is the most recent event received. While many widgets display an aggregation of all current events, each also highlights the current event using a consistent color. With this technique it is easy to see how the current event fits in to each display paradigm for extra cohesion.

These initial visualizations were developed in Java and Processing [23] and are targeted for autonomous gallery presentation instead of detailed user interaction. This means that the programs will run unattended for long periods of time and update without any requirement or opportunity for user input. This approach deviates from Ben Schneiderman's Visual Information-Seeking Mantra of "Overview, Zoom, Filter, Details on demand" [16] since we are not allowing users to interact directly with the visualization. Even so the design process took into account the overarching goal of Schneiderman's theory—ensuring the information is conveyed efficiently to those who want to see it. This puts extra pressure on the use of space and time to ensure it is interesting and not too cluttered since there is no opportunity for viewers to filter or modify the presentation.

With so many potential variables to investigate it was decided to segment the problem in to two separate but complementary tracks – **user-centric** and **content-centric**. When both visualizations are displayed simultaneously side-by-side they provide a detailed picture of real-time deleted tweet data.

4.1. User-centric Visualization

The user-centric approach seeks to understand deletions through the lens of the sender—who are they? What were they doing when they sent it? How much time elapsed before they deleted their tweet?

4.1.1. SparkTimeline

This widget is common to both visualizations. Data rates coming from the server are rendered as a continuously updating sparkline. Sparklines are described by Edward Tufte as "small, high resolution graphics embedded in a context of words, numbers, images" [20].

Time progresses to the right and the previous data points are plotted before the graph reaches the right side of the screen (the end of the display period), where it rolls over and starts from the left side again. The display period is variable and determines how fast the graph moves across the screen, as well as how dense the resulting lines appear. A period of 120 seconds is the default.

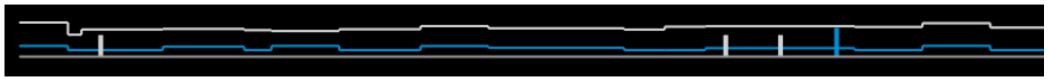
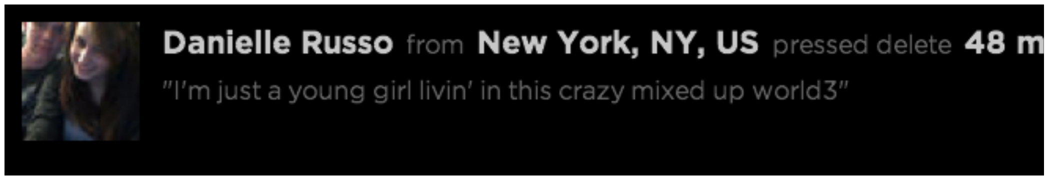


Figure 6: SparkTimeline widget

The white line depicts the rate of tweets being stored by the *Grab* stage and the colored line shows the deletes being checked by the *Retrieve* stage. All data points are graphed in terms of items per second. New events are marked on the timeline upon arrival as vertical lines. Previous events remain on the timeline until it rolls over.

4.1.2. UserDetails

This widget displays some basic user information and puts the deleted tweet in context. Metadata such as user screen name, location and number of followers helps build an image of who this person is in real life. The user's profile picture and biographical description are displayed to further humanize this person.



Danielle Russo from **New York, NY, US** pressed delete **48 m**

"I'm just a young girl livin' in this crazy mixed up world3"

Figure 7: Biographical Information in UserDetails widget

The deleted tweet is displayed in context with a few tweets sent around the same time that remain undeleted. The age of the tweet (how long ago it was originally sent) is also displayed. This reinforces the "lifetime" of the tweet since viewers can see how long ago this tweet was sent, and how far back in history the user had to go to find the delete button.

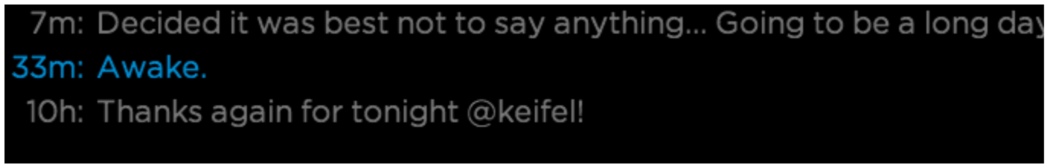

7m: Decided it was best not to say anything... Going to be a long da
33m: Awake.
10h: Thanks again for tonight @keifel!

Figure 8: Tweet context in UserDetails widget

4.1.3. Lifetime

The length of time that each tweet was publicly available is plotted as a single dimensional plot, with time progressing to the right. Time is apportioned along the axis using a logarithmic scale due to the large range of values involved.

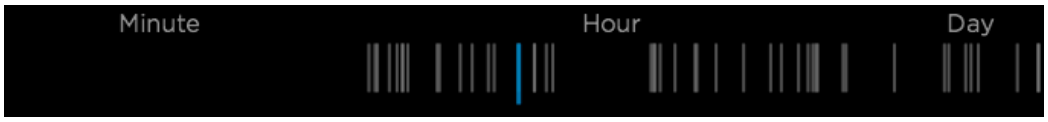


Figure 9: Lifetime widget

Labels are provided at meaningful times like Minute, Hour, Day and Week. Events are drawn as translucent lines (with the current event highlighted), allowing for relative density to be estimated where there are clusters of overlapping events.

4.1.4. Departure

The two primary events of a tweet's lifetime – being launched into the web and subsequently being snatched back – are modeled as a transit departure graph similar to those highlighted by Tufte [21]. Two days of time is displayed along a horizontal axis on top and bottom. Some events span more than one day, such as tweets that were sent in the evening and deleted the next morning. By displaying two days we can accurately place them on the timeline without the deleted time coming before the sent time. This follows the wrap around principle for schedules that Tufte discussed [22].

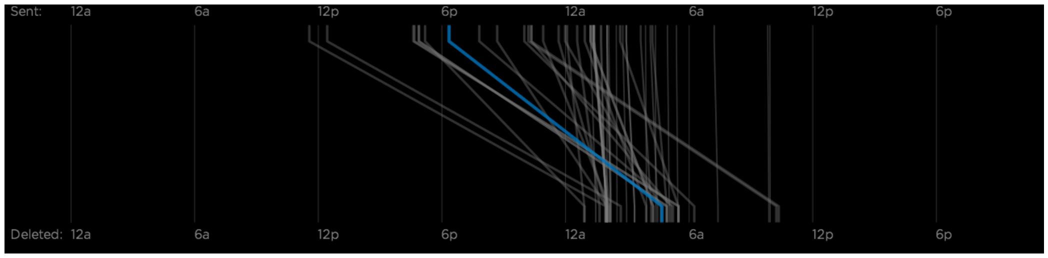


Figure 10: Departure widget

Each event is drawn as a line connecting the sent time on top to the time of deletion on the bottom. The graph primarily investigates the relative difference between time-of-day that a user sent their tweet and when they chose to take it back. This can show if tweets that were sent late at night are often deleted in the mornings.

4.2. Content-centric Visualization

The Content-centric visualization looks at the text of the tweet itself to investigate why it may have been deleted and how it compares to other deleted tweets. It originally included Natural Language Processing methods of content analysis but they were abandoned due to limited efficacy with the short, noisy twitter updates.

4.2.1. RecentText

The content of each tweet is the center of attention for this visualization, so the previous several tweets are displayed in rows with the current tweet highlighted. The user's profile picture is displayed next to each tweet to stamp it as unique – reminding viewers that unlike the rows of contextual tweets in the User visualization, each tweet comes from a different user.

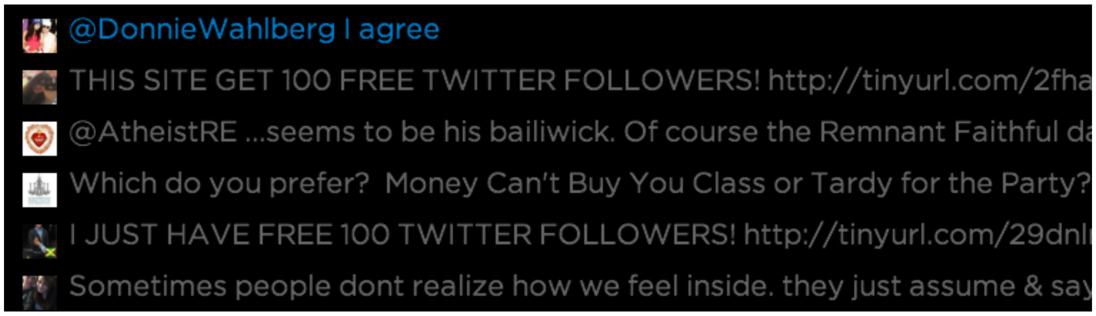

@DonnieWahlberg I agree
THIS SITE GET 100 FREE TWITTER FOLLOWERS! http://tinyurl.com/2fha
@AtheistRE ...seems to be his bailiwick. Of course the Remnant Faithful da
Which do you prefer? Money Can't Buy You Class or Tardy for the Party?
I JUST HAVE FREE 100 TWITTER FOLLOWERS! http://tinyurl.com/29dnl
Sometimes people dont realize how we feel inside. they just assume & say

Figure 11: RecentText widget

4.2.2. WordFrequency

This widget displays the most frequently seen words over the course of the visualization. Words in the current tweet that also appear in the most frequent words are highlighted which makes it easy to see how typical this tweet is. In this context, "words" are any tokens that are not URLs, emoticons or otherwise contain non-word punctuation. There is no requirement that tokens be actual words (since many are slang or abbreviations). These are entered into a RiConcorder from the RiTa library [27], which calculates cumulative word frequencies. Each time a new event arrives the word frequencies are updated and a new set of top words is extracted.

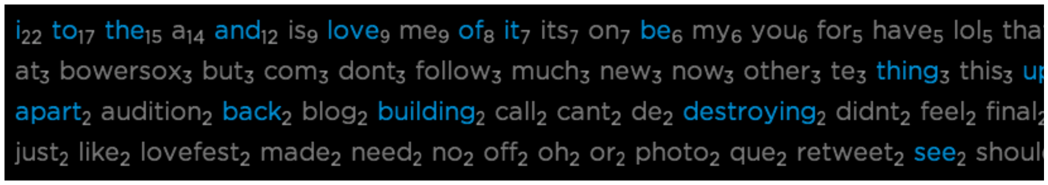
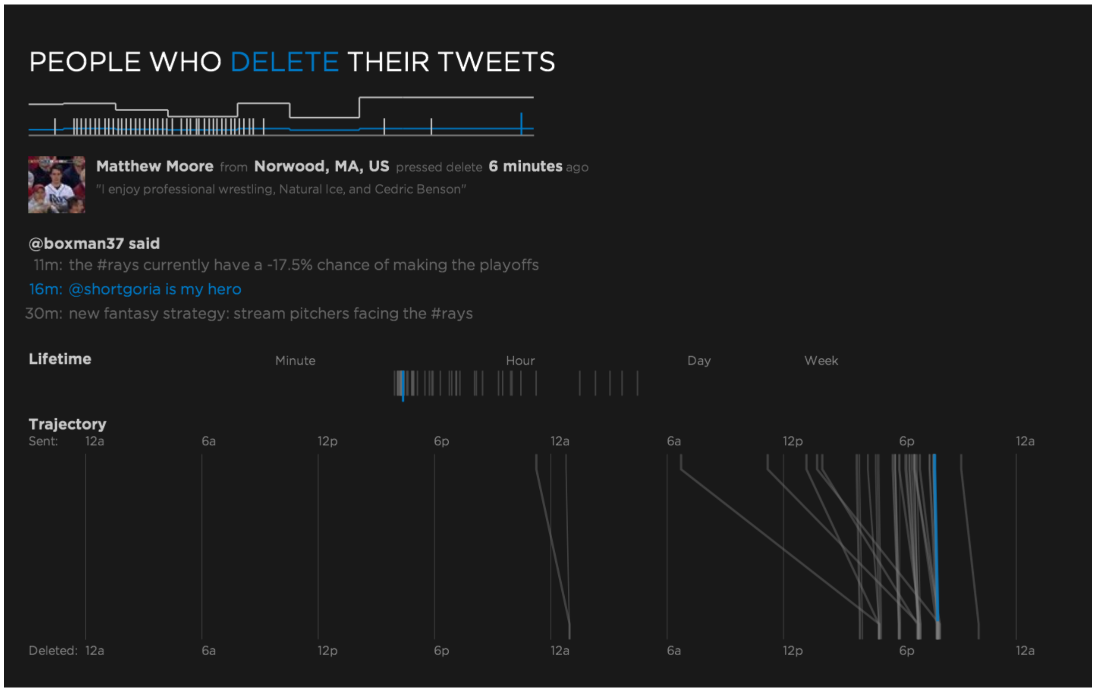

i$_{22}$ to$_{17}$ the$_{15}$ a$_{14}$ and$_{12}$ is$_{9}$ love$_{9}$ me$_{9}$ of$_{8}$ it$_{7}$ its$_{7}$ on$_{7}$ be$_{6}$ my$_{6}$ you$_{6}$ for$_{5}$ have$_{5}$ lol$_{5}$ tha
at$_{3}$ bowersox$_{3}$ but$_{3}$ com$_{3}$ dont$_{3}$ follow$_{3}$ much$_{3}$ new$_{3}$ now$_{3}$ other$_{3}$ te$_{3}$ thing$_{3}$ this$_{3}$ u
apart$_{2}$ audition$_{2}$ back$_{2}$ blog$_{2}$ building$_{2}$ call$_{2}$ cant$_{2}$ de$_{2}$ destroying$_{2}$ didnt$_{2}$ feel$_{2}$ final
just$_{2}$ like$_{2}$ lovefest$_{2}$ made$_{2}$ need$_{2}$ no$_{2}$ off$_{2}$ oh$_{2}$ or$_{2}$ photo$_{2}$ que$_{2}$ retweet$_{2}$ see$_{2}$ shoul

Figure 12: WordFrequency widget

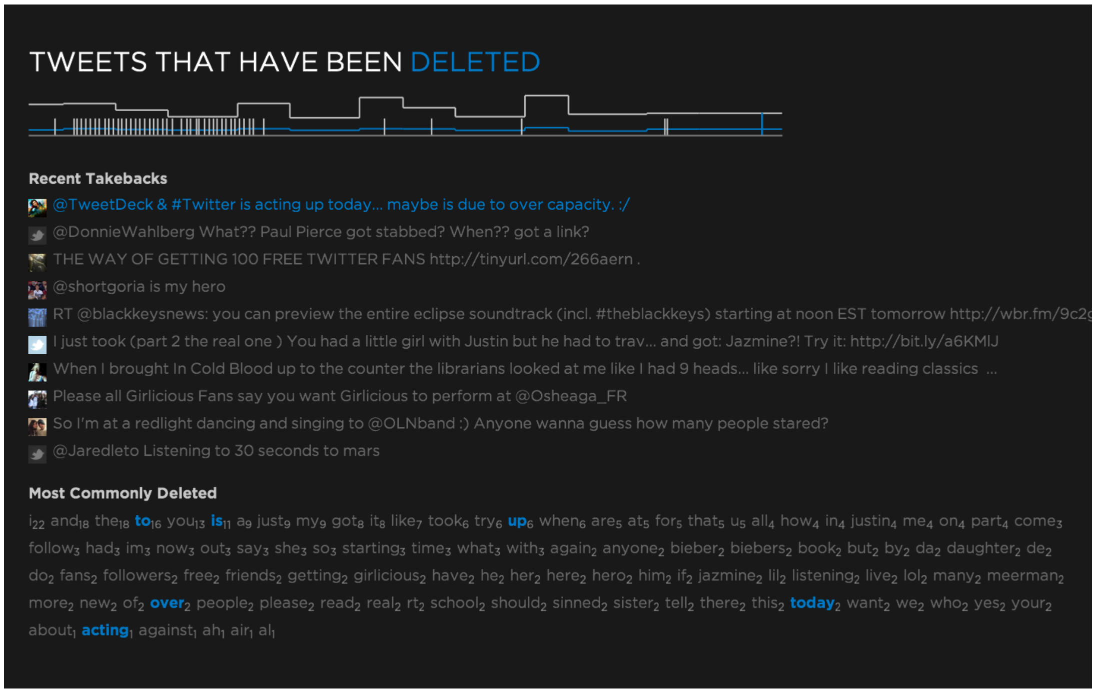


PEOPLE WHO DELETE THEIR TWEETS

Matthew Moore from **Norwood, MA, US** pressed delete **6 minutes** ago
"I enjoy professional wrestling, Natural Ice, and Cedric Benson"

@boxman37 said
11m: the #rays currently have a -17.5% chance of making the playoffs
16m: @shortgoria is my hero
30m: new fantasy strategy: stream pitchers facing the #rays

Figure 13: Full User-centric Visualization



TWEETS THAT HAVE BEEN DELETED

Recent Takebacks
@TweetDeck & #Twitter is acting up today... maybe is due to over capacity. :/
@DonnieWahlberg What?? Paul Pierce got stabbed? When?? got a link?
THE WAY OF GETTING 100 FREE TWITTER FANS http://tinyurl.com/266aern .
@shortgoria is my hero
RT @blackkeysnews: you can preview the entire eclipse soundtrack (incl. #theblackkeys) starting at noon EST tomorrow http://wbr.fm/9c2g
I just took (part 2 the real one) You had a little girl with Justin but he had to trav... and got: Jazmine?! Try it: http://bit.ly/a6KMIJ
When I brought In Cold Blood up to the counter the librarians looked at me like I had 9 heads... like sorry I like reading classics ...
Please all Girlicious Fans say you want Girlicious to perform at @Osheaga_FR
So I'm at a redlight dancing and singing to @OLNband :) Anyone wanna guess how many people stared?
@Jaredleto Listening to 30 seconds to mars

Most Commonly Deleted
i$_{22}$ and$_{18}$ the$_{18}$ **to**$_{16}$ you$_{13}$ **is**$_{11}$ a$_{9}$ just$_{9}$ my$_{9}$ got$_{8}$ it$_{8}$ like$_{7}$ took$_{6}$ try$_{6}$ **up**$_{6}$ when$_{6}$ are$_{5}$ at$_{5}$ for$_{5}$ that$_{5}$ u$_{5}$ all$_{4}$ how$_{4}$ in$_{4}$ justin$_{4}$ me$_{4}$ on$_{4}$ part$_{4}$ come$_{3}$ follow$_{3}$ had$_{3}$ im$_{3}$ now$_{3}$ out$_{3}$ say$_{3}$ she$_{3}$ so$_{3}$ starting$_{3}$ time$_{3}$ what$_{3}$ with$_{3}$ again$_{2}$ anyone$_{2}$ bieber$_{2}$ biebers$_{2}$ book$_{2}$ but$_{2}$ by$_{2}$ da$_{2}$ daughter$_{2}$ de$_{2}$ do$_{2}$ fans$_{2}$ followers$_{2}$ free$_{2}$ friends$_{2}$ getting$_{2}$ girlicious$_{2}$ have$_{2}$ he$_{2}$ her$_{2}$ here$_{2}$ hero$_{2}$ him$_{2}$ if$_{2}$ jazmine$_{2}$ lil$_{2}$ listening$_{2}$ live$_{2}$ lol$_{2}$ many$_{2}$ meerman$_{2}$ more$_{2}$ new$_{2}$ of$_{2}$ **over**$_{2}$ people$_{2}$ please$_{2}$ read$_{2}$ real$_{2}$ rt$_{2}$ school$_{2}$ should$_{2}$ sinned$_{2}$ sister$_{2}$ tell$_{2}$ there$_{2}$ this$_{2}$ **today**$_{2}$ want$_{2}$ we$_{2}$ who$_{2}$ yes$_{2}$ your$_{2}$ about$_{1}$ **acting**$_{1}$ against$_{1}$ ah$_{1}$ air$_{1}$ al$_{1}$

Figure 14: Full Content-centric Visualization

5. Discussion

5.1. The Nature of Internet Speech

Internet publishing may be considered in the vein of personal speech, especially in blogging and social network contexts. By taking our conversation online we are melding the boundaries between the traditionally formal grammatical structure of the written word and the more fluid nature of spoken language. Researchers have been closely observing this change in language, examining ways in which our online communications relate to what we are used to thinking about with language. Linguist David Crystal concluded that this new "netspeak" is "neither spoken language nor written language nor sign language, but a new language dimension of computer-mediated language" [31]. Twitter is an acceleration of this change in language since the input methods vary widely and the length of each update is rigidly limited, leading to many more people adopting the abbreviations and slang of netspeak. The result is that we compulsively shrink our words and even whole passages to fit into the space allowed (or the amount of typing we are comfortable with on our mobile device).

This textual shrinking has been accompanied by diminishing time spent composing each piece as well as less personal reflection time before sending it. A physical letter may take several minutes of careful thought to compose (avoiding misspellings that are difficult to correct), while an email may take just a minute or two. The time it takes to actually mail something provides additional reflection time, and many chances for those having second thoughts to retain the letter without ever sending it. Emails can similarly remain in a perpetual draft state, though the instantaneous "Send" button is always there, tempting us to simply dispatch it and move on to other things. Tweets represent the extreme of this trend towards quick publication without consideration–they require only a few seconds to write and “right now” theme of Twitter encourages us to just send it immediately so we can ready ourselves to send another. There are so many tweets being sent all the time it would seem quaint to agonize over individual details—setting us up to potentially regret things said in haste.

People are becoming more open with what they share online and yet simultaneously clinging to an expectation of privacy or stealth, dubbed the “privacy paradox” by Susan Barnes [17]. When things that were publicly viewable all along are suddenly thrust into the spotlight—such as when parents read their children’s online postings—the immediate reaction is often shock and surprise rather than responsibility for what was said. We forget that the Internet will never be a truly private space.

A topic of interest when considering Internet publication and the notion of responsibility is the question of single vs. multiple publication. Is mass-publication defined by a single event (such as hitting the “submit” button on a blog post), or is the act replayed every time someone else views the work? This can be a big distinction on the Internet when items can be viewed by thousands of people, and it is not always possible to get an exact count.

This question has roots in disputes of defamation with the very first printed publications. When assessing damages, courts had to consider if the publisher was liable for just the initial act of publication (regardless of the number of copies produced) or could be assessed for each instance of the printed work (perhaps thousands of times). The current legal precedent has converged on the *single publication rule*, which stipulates that a single *edition* of something can make the publisher liable for only one case of libel [31]. There is still debate as to how to map the concept of an edition to the digital realm.

Though primarily a legal concept, the single publication rule can influence how we think of deleted tweets. It may caution us to avoid ascribing too much importance to tweets that were deleted after a long time. Though the tweet may have been viewable for days or even weeks, a user trying to rescind their speech out of fear of causing offense is no more liable than if they had deleted it right away.

Since Twitter is often described as a form of "microblogging" it is important to consider how even normal-sized blogs are different from traditional publishing. "Weblogs" have become quite popular in part because of the low barrier to entry—blogging platforms are usually free and easy for novices to use. Writers are expected to "be themselves" and not conform to rigid standards of content and refinement. An interesting characteristic of blogs is that entries are arranged chronologically and not alphabetically or in a logical order that supports an overall argument [27]. Twitter takes this point to the extreme as several consecutive tweets by one person may all belong to different conversational threads or may be independent expressions. This limits the amount of automatic contextual interpretation we can do based on the data presented in the visualization – only other humans can really decide how a tweet fits into the surrounding context, if it fits at all.

5.2. Analyzing Deleted Tweets

Information takes on a new dimension when it has been deleted. Previously mundane statements suddenly acquire newfound salience and provoke many questions, the foremost usually *why did they delete this? Aether* does not purport to definitively and automatically decide why something is deleted – that is both computationally hard and prone to error since only the sender can truly say why something was deleted. Instead it seeks to make information available to let viewers muse on the motivations of the sender—both when they sent the tweet and when they changed their mind. In doing so many other questions can be explored along the way, such as *who is this person*, and *how were they using Twitter?*

Twitter can seem impenetrable to some digital humanists and researchers, offering an avalanche of enticing human data while at the same time defying the application of many time-honored analysis methods. Tweets seem too short in length for *close reading* (the painstaking analysis of a single piece) and too disjoint for true *distant reading* (Franco Moretti's idea of poring over large datasets to gain insight through aggregation [34]). The text seems at once noisy and heavily encoded—typos and sentence fragments co-exist with emoticons, URLs and other semantic elements employed to pack as much meaning into 140 characters as possible. Twitter users' fluid grammar and hit-or-miss spelling thwarts many efforts at parts-of-speech tagging and classification. Social datasets like Twitter are likely to only grow in size and ubiquity, leaving much work for digital humanists and computer science researchers.

While computational tools have a ways to go before they can be reliably applied to random sets of tweets, there are other angles that can be examined at present. Placing the rescinded tweet in context with ones sent immediately prior and afterward helps envision the users' behavior patterns. Though many tweets contain typos, it is easy to see if that was why it was deleted. *Aether* estimates this by comparing each tweet to the collection of other tweets sent by the user around that time (indicating that the user re-sent the tweet with only minor corrections). Incidentally, out of a sample of 20,000 deleted tweets, only 16% were found to be typos using these criteria.

Specific dates can be correlated with events in the news, or days of the week (do people spend Monday complaining about the new week, only to have second thoughts and delete the later in the week?). Time of day can be examined as well such as if tweets sent on Friday nights out are often deleted Saturday mornings.

Time continues to play a central role with the consideration of the tweet lifetime (the number of seconds that elapsed between when the tweet was sent and when it was deleted). During this time the user was doing other things while their tweet floated through cyberspace being seen by an unknowable number of entities. When plotted in the Departure widget of the User visualization it is clear that many tweets are deleted within an hour of being sent. And yet, there are ones that were deleted days, even weeks later. It took effort for the user to even find the tweet in the Twitter interface in order to hit the delete button, and it is likely that no human twitter users were looking at that tweet anymore. These long-tail deletions offer one of the most interesting points of investigation in *Aether*.

Though machine-learning approaches to content categorization were not successful (see the discussion of Natural Language Processing tools in 5.3), some basic classification in *Aether* was still possible using regular expressions. Dividing content into categories can provide a new angle to the data, exploring questions such as if explicit tweets are deleted more often than tweets with non-offensive language. Regardless of the efficacy of classification and the amount of meaningful categories employed there will always be some tweets that can only be labeled as idle chatter. This chatter takes the form of such vacuous statements as “whats up?” and “Awake.” Both are real tweets that were really deleted. Why would someone bother to delete a short snippet of conversation that may not have had a purpose to begin with? That question alone may stump the viewers of the system.

Recent legal cases also suggest that some deletions may be motivated out of self-preservation. There are always media reports of people getting in trouble as the result of immodesty online --- such as losing their job due to scandalous postings on Facebook. Recently tweets have started becoming used as evidence in court – to the chagrin of their senders. The case of Amanda Bonnen vs. Horizon Group is an example that demonstrated both the use of Twitter in the courtroom, and the risk that corporations take in using engaging single people through the legal system [36]. In this case Bonnen used Twitter to complain about mold in her Horizon Group-owned apartment. Horizon sued her for defamation, promptly elevating the case to national attention. Bonnen was a small-scale Twitter user with less than a few dozen followers who saw her original tweet. The tweet in question ended up being read by millions as a result of media publicity, giving substantial negative publicity to Horizon. In the end the court dismissed the case, citing the tweet as being “too vague to meet the strict definition of libel.” While this episode ended happily for the defendant it is entirely plausible that people may have second thoughts about a pointed comment they made and take it back in hopes of avoiding litigation or other negative consequences.

The autonomous *Aether* system is not the only actor in this equation. Like any software it is unfeeling and does not care about the semantic and emotional content of other peoples' tweets; it just calmly collects them as directed. The viewers play a role of interpreter since they may empathy towards those whose thoughts and feelings are being sampled and dissected. When we see these statuses on the wall that were taken back we feel a sense of voyeurism, like we are spying on them.

Some viewers of the installation have questioned the legality or ethicalness of storing and presenting other users' deleted tweets. While the moral and ethical considerations surrounding this process are an enduring question (and provide purpose to this project), the legality can be addressed by looking at the Terms of Service and other documentation published by Twitter. The primary Terms of Service (aimed at general users of Twitter) does not cover deleted tweets [39]. The *Developers Rules of the Road* specifically mentions deletion saying:

> Respect the privacy and sharing settings of Twitter Content. Promptly change your treatment of Twitter Content (for example, deletions, modifications, and sharing options) as changes are reported through the Twitter API [38].

Though not clearly spelled out, this would hint that Twitter does not want developers publicly displaying tweets that have been deleted. Twitter's help pages mention that while you can delete your tweets from their system, they may remain in search indexes [17] and other site language also warns that it may remain in third party applications (such as this one). Future work includes an idea to operate a "lost and found" that catches these tweets and sends them back to the person that deleted them. A compelling element to this plan is the unknown level of user response or backlash. Users reacting strongly to such a system may in fact cause Twitter to tighten and enforce their policies with regards to the use of deletion requests.

5.3. Natural Language Processing for Twitter Data

Natural Language Processing methods were investigated to further analyze the content of the tweets but were largely abandoned due to inefficacy. Twitter updates are difficult to process using higher-level tools because each individual update is short (less than 140 characters) and is low in content.

Automatic content-based categorization and classification was investigated to shed light on the nature of the tweet's content. These tools typically require assembling a large training set of data for each possible category. These sets are used to generate a 'fingerprint' that characterizes the items that are expected to fall into each category. New items can then be categorized by comparing their contents to each fingerprint using statistical tests. Some researchers have found success trying to classify tweets based on content and sentiment using a variety of techniques [39] [40]. The Java Text Categorization Library [41] was used to experiment with text classification for *Aether* but results were not reliable due to the limited size of input text.

Parts-of-Speech (POS) tagging involves labeling each word in a sentence with its linguistic part of speech (such as noun or verb). Digital humanists use POS tagging to investigate individual sentence structure and dialect as well as inform research in linguistic trends. POS tagging efforts with tweet data were mildly successful but ran into contextual inaccuracies. Words that could be a noun or a verb were often mislabeled, an error that can be avoided with contextual awareness not easily available with such limited, noisy data. Many tokens found in tweets do not readily map onto our notion of part-of-speech, such as emoticons and hashtags. The RiTa library [30] was used to explore POS tagging since it is designed for Processing and has many different text processing features.

One interesting NLP tool that deserves further study is the Markov-chain sentence generator. This works by analyzing a large group of sentences to produce a statistical model that can be used to generate similarly structured sentences. While it can be difficult for these tools to generate truly realistic sentences, a cursory investigation suggested that they are well suited for use with Twitter content. Since tweets are sentence-like in length and structure but typos and grammatical issues are tolerated, the typical shortcomings of a Markov-chain generator are accommodated.

Generated pseudo-tweets appeared to be very similar to actual tweets. This could be used in a future work that looks at the fluidity of language evidenced in status updates, and how possible it is to fool people with fake updates.

5.4. Design Assumptions

There are many decisions and assumptions made during the design of this system that affect its current architecture and operation.

The decision to target real-time visualization (as opposed to offline batch processing) has had the greatest influence on the design of the system and is the primary reason the server is divided into separate processes since it is so important to optimize the throughput of each stage. This factor also influenced issues of memory management since there is not an upper bound on the number of items that will be processed, or on the total runtime of the software system.

The initial visualizations are targeted at autonomous gallery display, which affected how they were designed. Without the ability for user interaction and filtering the widgets must be carefully laid out to maximize data presentation while minimizing clutter. It also encouraged more use of time in the displays to reinforce the dynamic nature of the data since people will not be directly using the system to see new data come in. By saying that something happened “less than one minute ago” and updating the age as time elapses it is more obvious if data is fresh and new.

The primary data gathering assumptions involve heuristics to discard as much data as possible to keep the workload manageable. As mentioned previously, each tweet is checked against a configurable skip function in the *Grab* stage before it can be saved. Those that fail the test are immediately discarded. The skip criteria may have the biggest effect on the character of the system and as such must be carefully crafted to segment the population of tweets in a logical, well-defined manner and reject those that would negatively impact the output.

It was decided that the skip tests for *Aether* would be designed to limit the acquired data to tweets sent by users in the United States or Canada, and written in English. This drastically cuts down on the number of tweets that end up being saved while still keeping a coherent grouping (instead of discarding them randomly). The written language requirement was developed since the audience viewing the visualizations will be predominantly English speakers.

Country of origin is determined via the user profile's location as well as UTC offset (time zone). Both of these metrics have the potential to be misleading since they are user-supplied via the Twitter settings page. Relying on the estimated location is fraught with peril since the location string is free-text, may not be their actual location, and may not even be a real place. Since geocoding takes non-negligible time it is relegated to post-retrieval and cannot be used to skip a tweet outright before saving, thus wasting memory. The UTC offset is a more effective test since it is quick to check and it is probable that most users have filled it in correctly since this setting affects how twitter displays the times of all the tweets on their website.

Written language is checked by looking at the user profile "language" field. This too can be prone to errors as not all users have picked the right language in their Twitter preferences, and some people write in more than one language. A Ruby library called 'whatlanguage' is used in the *Retrieve* stage to estimate the language based on the actual textual content.

5.5. Technical Performance Considerations and Results

The *Aether* server software was developed and tested on a dedicated Ubuntu Linux server with a quad-core 2.5Ghz Pentium processor and 2GB of RAM. It was given a publicly accessible hostname, which simplified client-server access. Clients were run on a variety of computers and networks.

Unrecovered tweets were originally archived in a regular MySQL database but the insert bandwidth was found to be a primary performance concern. Though MySQL is a relatively fast database system the test server was not powerful enough to handle a never-ending high volume stream of insertions while also performing the rest of *Aether*'s functions, such as interacting with clients. The buffers between *Grab* and the database-storing stage constantly overflowed, causing up to 50% of all data to be lost. Additionally, Twitter throttled the amount of data being sent since it was not being consumed fast enough. This resulted in a far lower deleted tweet recovery rate while still using almost all of the server's resources and negatively impacting the other *Aether* components.

These MySQL performance issues resulted in making a switch to Memcached [38], a technology originally developed by *Livejournal* as a distributed caching system to increase performance of high traffic websites. Memcached stores all data in volatile RAM memory instead of on disk so it is not *persistent,* meaning the data disappears if the Memcached or the computer is restarted. This is contrasted with regular databases such as MySQL, which maintain data until it is intentionally deleted. Memcached instances on several computers can be joined together to provide a larger storage space and potentially increased performance as the load is shared across machines. It was decided that not losing any data when it arrives from Twitter is a better short-term goal than having an infinitely large archive but not knowing how much data was never saved.

Unrecovered tweets are currently being held in a 768MB Memcached instance running on the test server. In practice only about 88% of the capacity is available for data items—the rest is used by Memcached for recordkeeping and other tasks. As the cache fills up, older tweets get evicted which limits how far back in time *Aether* can look. In practice, the limit is about 500,000 tweets, covering about 12 hours of history assuming a nominal saved tweet rate of 12 per second. Increasing the size of the local Memcached instance or expanding the pool to include Memcached instances on other servers would increase the maximum look-back time.

While data rates fluctuate, about 4% of all data items received were deletion requests, the rest were new tweets. On average about 5% of these deletion requests were successfully retrieved. The other 95% of missing tweets can be explained by a few factors. First, we discard up to 60% of all tweets received in order to save resources and limit the data to the desired demographics so these tweets will never be recovered. As mentioned previously, increasing the size of the Memcached instance or using a different database system would increase the number of tweets that can be archived at once, increasing the recovery rate.

Besides technical implementation concerns, there are a few known systemic limitations to the current approach. *Aether* may not suitable for academic statistical analysis since the sample stream contains only 5% of all tweets sent and we discard a large percentage of them to save resources. Additionally, the design decision of pursuing *real-time* analysis effectively prohibits doing heavy processing on every single tweet received (such as geocoding). This makes it difficult to compare recovered tweets to the larger population since unrecovered tweets do not have as much derived metadata.

5.6. Future Work

Most web services today provide a standard HTTP / REST API, which allows a variety of different uses for the data including access by a standard web browser. *We Feel Fine* is a perfect example of an information art project that added an API to encourage the use of its data. At present there is no such interface but one could easily be added to complement the existing OSC API since the data is stored in a standard database. It would function as an application separate from *Aether* that retrieves deletion Event information directly from the database and returns it to clients via a web server such as Apache. These APIs tend to be more discrete and history-oriented (sending back finite chunks of previously existing data) but with additional infrastructure a real-time streaming system similar to the current Twitter Stream API could be produced. *Aether* would then begin to act like a sort of filter for the main Twitter Stream API, providing a subset of the main Twitter stream to interested clients.

It would be useful to revisit MySQL or investigate new database solutions to replace or augment the current Memcached archive. Disk-based persistent databases would dramatically increase the recovery rate since tweets can be archived for a longer time. Changing or adding databases is only feasible of the insert performance issues can be addressed in order to not lose data.

There are many other applications of the deleted tweet data besides ephemeral visualization. A Twitter account named @tweet_morgue has been created to collect these “dead” tweets and make them available for others to see using Twitter’s own infrastructure. If the dead tweets are prefixed with the original sender’s twitter screen name (a mention) then the sender will likely see their deleted tweet come back onto their timeline, provoking interesting user reactions. By sending deleted tweets back to the originator the @tweet_morgue also acts as a sort of lost & found for errant tweets. A primary theme of this project so far is that the senders do not realize they are being observed. Would their behavior change if they were made aware that their deletions are being archived and dissected?

6. Conclusion

Aether has proven to be a robust platform for real-time processing and recovery of deleted tweets. The multi-stage pipeline approach and separated client-server architecture has allowed for a flexible system that can support several heterogeneous clients running simultaneously at multiple locations. The push-style client communication over OSC allows for clients to display deleted tweet information within seconds of the real-world deletion event.

The data produced has provided an interesting window into an often-ignored part of the data lifecycle. Initial visualizations made the data accessible to the general public and often provoked questions and observations as people viewed the previously hidden actions of Twitter users around the country.

Future work will continue to make the data available in more formats and to more people. As people realize their online data is not completely in their control they may reconsider what they say and give up deleting all together. Until then, *Aether* will be watching.

References

[1] Lisa Jevbratt. (2006, June) Searching traces of We: Mapping Unintended Collectives. [Online]. http://jevbratt.com/the_voice/the_voice_cat.pdf

[2] Jonathan Harris and Sep Kamvar. (2005, August) We Feel Fine. [Online]. http://www.wefeelfine.org

[3] Kevin Hansen. (2007) Secret Regrets. [Online]. http://secondchanceonline.blogspot.com/

[4] Frank Warren. (2005, January) Post Secret. [Online]. http://www.postsecret.com

[5] Jane Mulfinger and Graham Budgett. (2005) REGRETS. [Online]. http://regrets.org.uk/

[6] A. Java and X. Song, "Why we twitter: understanding microblogging usage and communities," in *WebKDD Workshop on Web Mining*, San Jose, CA, 2007, pp. 56-65.

[7] Bernardo Huberman, Daniel Romero, and Fang Wu, "Social Networks that matter: Twitter under the microscope," 2008.

[8] Courtenay Honeycutt and Susan C. Herring, "Beyond Microblogging: Conversation and Collaboration via Twitter," in *42nd Hawaii International Conference on System Sciences*, Hawaii, 2009, pp. 1-10.

[9] Internet Archive. (1996) Wayback Machine. [Online]. http://www.archive.org/web/web.php

[10] Steve Lohr, "Library of Congress Will Save Tweets," *New York Times*, Arpil 2010.

[11] Tom Scott. (2009) Tweleted. [Online]. http://tweleted.com

[12] Inc Yahoo. (2010) Yahoo! Pipes. [Online]. http://pipes.yahoo.com

[13] IBM Visual Communication Lab. (2004) Manyeyes. [Online]. http://manyeyes.alphaworks.ibm.com

[14] Angus Forbes, "Behaviorism," Santa Barbara, CA, MS Thesis 2009.

[15] Alan Newberger and Jeffery Heer. (2007) Prefuse. [Online]. http://prefuse.org

[16] Twitter. (2010) Twitter API Wiki. [Online]. http://apiwiki.twitter.com/Streaming-API-Documentation

[17] Martin Hepp. (2010, February) Hyper Twitter: Weaving a web of linked data, tweet by tweet. [Online]. http://semantictwitter.appspot.com/

[18] Teng-Sheng Moh and Murmann Alexander J., "Can You Judge a Man by His Friends? - Enhancing Spammer Detection on the Twitter Microblogging Platform Using Friends and Followers," in *Information Systems, Technology and Management*, vol. 4, Bangkok, 2010.

[19] Wikipedia Contributors. (2010, May) Levenshtein Distance. [Online]. http://en.wikipedia.org/w/index.php?title=Levenshtein_distance

[20] Ruby Community. (2010) Ruby Programming Language. [Online]. http://ruby-lang.org

[21] Masatoshi Seki. (2003) Distributed Ruby Documentation. [Online]. http://www.ruby-doc.org/stdlib/libdoc/drb/rdoc/index.html

[22] UC Berkeley CNMAT. (2002) Open Sound Control. [Online]. http://opensoundcontrol.org

[23] Bryan Ford, Pyda Srisuresh, and Dan Kegel, "Peer-to-Peer Communication Across Network Address Translators," in *USENIX Annual Technical Conference 2005*, 2005.

[24] SBBI. (2006, Nov.) UPnPlib. [Online]. http://www.sbbi.net/site/upnp/index.html

[25] George Legrady. (2005) Making Visible the Invisible. [Online]. http://www.mat.ucsb.edu/~g.legrady/glWeb/Projects/spl/spl.html

[26] Antony Unwin, "Interacting With Graphics," in *Graphics of Large Datasets*.: Springer, 2006, p. 77.

[27] Casey Reas and Ben Fry. (2010) Processing. [Online]. http://processing.org

[28] Ben Schneiderman, "The Eyes Have It: a Task by Data Type Taxonomy for Information Visualizations," in *Proceedings of the 1996 IEEE Symposium on Visual Languages*, College Park, MD, 1996, p. 336.

[29] Edward Tufte, *Beautiful Evidence*., 2006.

[30] Edward Tufte, *Envisioning Information*.: Graphics Press, 1990.

[31] Edward Tufte, "Theory of Data Graphics," in *Visual Display of Quantitative Information*, 2nd ed.: Graphics Press, 2001, p. 98.

[32] Daniel C. Howe. (2010) RiTa. [Online]. http://www.rednoise.org/rita/

[33] David Crystal, *Language and the Internet*.: Cambridge University Press, 2001.

[34] Susan B. Barnes. (2006, September) A Privacy Paradox: Social Networking in the United States. [Online]. http://firstmonday.org/issues/issue11_9/barnes/index.html

[35] Sapna Kumar, "Website Libel and the Single Publication Rule," *The University of Chicago Law Review*, no. Spring, pp. 639-662, 2003. [Online]. http://www.jstor.org/stable/1600592

[36] Torill Mortensen and Jill Walker, "Blogging Thoughts: Personal Publication as an Online Research Tool," in *Researching ICTs in Context*, Andrew Morrison, Ed.: InterMedia Report, 2002, pp. 267-279.

[37] Franco Moretti, *Graphs, Maps, Trees: abstract models for a literary history*. London: Verso, 2007.

[38] Wikipedia Contributors. (2010, May) Horizon Group v. Bonnen. [Online]. http://en.wikipedia.org/wiki/Horizon_Group_v._Bonnen

[39] Twitter, Inc. (2009, September) Twitter Terms of Service. [Online]. http://twitter.com/tos

[40] Twitter. (2010) Developer Rules of the Road. [Online]. http://dev.twitter.com/pages/api_terms

[41] Twitter, Inc. (2009, January) How to Delete a Tweet. [Online]. http://help.twitter.com/forums/10711/entries/18906

[42] Alec Go, Richa Bhayani, and Lei Huang, "Twitter Sentiment Classification using Distant Supervision," Stanford University, Paper 2009.

[43] Microsoft Research. (2010) Twahpic. [Online]. http://twahpic.cloudapp.net/S4.aspx

[44] Knallgrau New Media Soultions. (2006) Java Text Categorization Library. [Online]. http://textcat.sourceforge.net/

[45] Memcached team. (2010) Memcached. [Online]. http://www.memcached.org/

Appendix

A) Twitter Data

Below is an example of data that Twitter sends for each tweet in the streaming API, in JSON format. It includes the nested User object as well as an example of the geotagging feature which is not yet widely-adopted by Twitter users.

```
{
    "contributors": null,
    "created_at": "Thu Apr 29 22:53:35 +0000 2010",
    "source": "<a href=\"http://ubertwitter.com\"
rel=\"nofollow\">UberTwitter</a>",
    "in_reply_to_status_id": null,
    "place": null,
    "geo": {
      "type": "Point",
      "coordinates": [
        56.419056,
        -3.40316
      ]
    },
    "in_reply_to_screen_name": null,
    "user": {
      "profile_background_tile": false,
      "friends_count": 127,
      "description": "",
      "lang": "en",
      "favourites_count": 26,
      "verified": false,
      "created_at": "Thu Nov 20 02:36:19 +0000 2008",
      "profile_background_color": "9ae4e8",
      "following": null,
      "profile_text_color": "000000",
      "url": null,
      "statuses_count": 7038,
      "time_zone": "Edinburgh",
      "profile_link_color": "0000ff",
      "profile_image_url":
"http://a1.twimg.com/profile_images/456261026/twitterProfilePhoto_normal.jp
g",
      "geo_enabled": true,
      "notifications": null,
      "followers_count": 63,
      "protected": false,
      "location": "Anywhere you are!!",
      "contributors_enabled": false,
      "profile_sidebar_fill_color": "e0ff92",
      "screen_name": "fifinoir",
      "name": "fiona",
      "profile_background_image_url":
"http://s.twimg.com/a/1272044617/images/themes/theme1/bg.png",
      "id": 17502304,
      "utc_offset": 0,
      "profile_sidebar_border_color": "87bc44"
    },
    "in_reply_to_user_id": null,
    "coordinates": {
      "type": "Point",
      "coordinates": [
        -3.40316,
        56.419056
      ]
    },
    "truncated": false,
    "id": 13094530000,
    "favorited": false,
    "text": "Ok back not getting too bad. Hopefully I'll get some sleep."
  }
```

B) Aether Data

B.1) Tweet

Every tweet received is stored as its own object along with a small subset of user information to make retrieval easier.

Identifier:	Tweet ID User ID
User details:	Tweet source (program used to send the tweet)
Time:	Time sent, time deleted in UTC User UTC offset for adjusting to local time Lifetime (number of seconds before tweet was deleted)

B.2) User

User objects contain a subset of the fields that Twitter sends with each tweet. Users are stored separately to track how many times each person has deleted a status.

Identifier:	Twitter user account ID
User details:	Screen name (account name) Display name (supposedly their real name) Location string (not geocoded) Biographical description Profile image URL
Time:	UTC offset in seconds Created_at (time registered)
Counts:	Friends, Followers count Statuses count Count of deleted tweets from this user (that we know of)

B.3) Event

Events are created from a recovered deleted tweet. They contain a mixture of official Tweet and User data as well as derived metrics.

Identifiers:	Tweet ID User ID
Text:	Original text “Word text” – original text, minus punctuation and non-word tokens (URLs, hashtags, etc) “typo text” –The tweet that replaced this one if this is a typo Extracted Hashtags, URLs, Mentions
Time:	Time sent, time deleted in UTC

	User UTC offset for adjusting to local time Lifetime (number of seconds before tweet was deleted)
User details:	Screen name Biographical description Location string Geocoded location (latitude & longitude) Tweet source (program used to send the tweet) Profile image URL
Counts:	Friends, Followers count Count of deleted tweets from this user (that we know of)
Other tweets:	A set of tweets sent before and after the deleted tweet, retrieved at recovery time. Useful for building context.

C) Aether OSC API

All communication between clients and the *Aether* server takes the form of OSC message passing. Sending a message to an address on the server will usually result in the server sending one or more messages back to the client. Except for initial registration, the client must include the unique client 'key' with each message for verification.

C.1) Server side (messages from client)

/client/register [IP] [port]	Initial registration – tell server where client is listening.
/client/deregister [key]	Goodbye message – tell server to stop communication.
/client/hello [key]	Periodic keep-alive message to ensure reliable communication. Should be sent at least once per minute.
/event/latest [key]	Request the most recent event.
/event/random [key]	Request a random event from the recent history buffer.
/event/history [key]	Request full history buffer (50+ events).
/rate [key]	Request most recent datarate.

C.2) Client side (messages from server)

/client/register/id [key]	Server confirms registration & sends unique client key.
/event/latest [event json]	A specific non-history event.
/event/history/start [expected n]	Signify the beginning of history messages. Include the expected number of messages.
/event/history/event [event json]	One event from the history buffer.
/event/history/end [actual n]	Signify that all history messages are sent, with count of actual events successfully sent.
/rate [rate json]	A hash of data rates, sent frequently.