Navigating Articles of News Websites with a Web Metrics Visualization

Alain Ibrahim

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

March 2015

1 Abstract

Current prominent web traffic analytics systems such as Google Analytics log, report and visualize traffic and other data about websites. They also allow filtering the views of this traffic through graphical controls. These analytics systems are geared to website administrators who use the data to identify patterns and assist management in strategy making. For news websites, administrators analyze the traffic on a news article as well as the geographic locations of the article's online visitors. The visitors, however, will at best see metrics such as Facebook likes, number of shares and tweets, and a "top-read" list in a corner inside the visited article's page. In addition, the running list of articles on the home page is solely determined by the news web editors; thus, current news website visitors do not have a direct view on the short term trends of all published articles. In order to add this capability for the visitors, I built a system that hooks into a news website, captures metrics about access frequency, times and locations, and conveys the near real-time traffic through an interactive web-based visualization.

Contents

2 Thesis Project Description

As of 2015, the online media web can be divided into two categories: news websites that contain stories about world events, and social media sites that contain conversations centered around these events. Before social media became ubiquitous, the news information flow was unidirectional, with hardly any feedback from the website visitors. Later, visitors started sharing links on their online social pages such as Facebook and Twitter, creating a secondary platform to promote news articles. Though, what appears on the home page of one's favorite news website is set by the web news editorial team and can be influenced by the news station's editorial or business agenda. For example, a breaking news story about the death of a famous actor can overshadow another breaking story about a medical discovery that helps cure diabetes. In reality, the latter story ends up having less exposure and can be missed by medicine enthusiasts.

In this thesis, I aim to create a supplementary method of showcasing the news articles, using an interactive web visualization. The product is geared to website visitors that are curious in knowing the duration of an article's popularity and the interest level of the readers when they read that article. The website visitors should be able to simultaneously compare theses measures across two articles.

News stations use different web content management systems (WCMS). The system that I built hooks into a very common and open source WCMS called Wordpress. It then captures web metrics, and visualizes these met-

rics inside an interactive web-based visualization. The visualization targets website visitors but can be equally used by web admins. There are three types of data I took interest in capturing: The number of hits of an article, the amount of time spent reading an article, and the geographic locations of the visitors.

In the first stage, I created a database to house the web metrics that I wanted to capture. I then created a server-side library that can be included into any Wordpress-based website. This library hooks into my database and captures the hits' metadata against the given news site. Such metadata includes time of article visit, region of visitor, and timezone of visitor. I then built the "business logic" of the data and information that I wanted to capture. This includes algorithms that produced the refined data and the metrics to be ultimately communicated to the end user. This amounted to the final step of creating an interactive visualization that shows near real-time user web traffic in each of the registered articles of the connected news site.

The domain that I targeted is **online news media**. I captured the below data:

- The number of visitor hits to a given article. Each hit is defined by a unique timestamp and IP address.
- The visitors' geographic locations.
- The amount of time visitors spent reading a specific news article.

The interactive visualization enables four tasks:

1. **To compare articles' hits and reader attention spans for any period within the last month, given a select category.**
2. **To preview each article's thumbnail, title, and excerpt.**
3. **To compare the geographic distribution of hits and attention of up to two articles.**
4. **To navigate to the full article on the news website.**

As for interactivity, the visualization enables the following:

- Hovering over article elements to preview the thumbnail, title, and excerpt of the body content.
- Clicking on an article representation to display a table that shows the geographic distribution of the number of hits and attention span for that article.
- Filtering the data via a controls box. The controls box allows the user to select a desired news category, to filter by date range, and to toggle context-related settings.

The final product is geared towards the conventional news site structure, where categories are assigned to categories (e.g. sports, health, science, and politics).

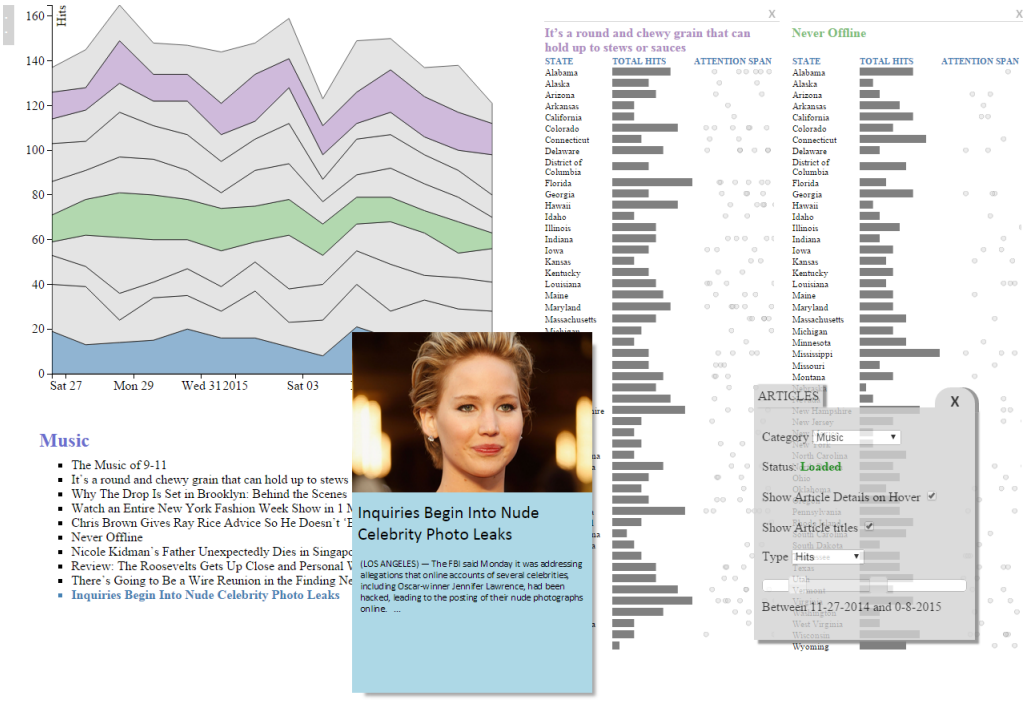


Figure 1 – The user previews a music article in the interactive visualization while having selected two articles for hits and attention span comparison.

3 Prior Work

3.1 Visualization

Prominent researchers in the field of visualization have concocted logical constructs and approaches to designing and implementing quality visualizations. From a presentation standpoint, there is much emphasis on showing the end user just enough visual data for the user to achieve the intended goals of the visualization — this is known as **expressiveness** (Schumann & Muller, 2000). Another popular concept in visualizing efficiently is the **data-to-ink ratio** (Tufte, 2001). The idea is to have just enough ink presented to convey the data; additional ink that is not matched by data is a distraction.

Having an efficient visualization is not enough. It needs to be effective. This brings us to two concepts covered in depth by Tamara Munzner: **domain** and **task** (Munzner, 2009). She advocates that in the design phase, the builder needs to identify the domain that he/she is working with. A domain is a world of ideas and concepts that are abstracted under one umbrella. For example, human anatomy, software engineering, economics, and biology are all domains. A domain gives rise to concepts, definitions, and a pre-built library of shapes and colors by the convention of collective knowledge.

On the other hand, the task refers to the "what" of the visualization. What is/are the goal(s) of the visualization? Is it to explore, analyze, extrapo-

late, etc.? Clearly identifying the tasks of the visualization at hand solidifies the direction in which the designer uses the visual variables to encode data (Carpendale, 2003). Encoding refers to the idea of visually representing a quantitative or qualitative data point. The visual variables include position, size, shape, value, color, orientation, grain, and texture. For an effective visualization, the choice of encoding is dependent on the domain of the visualization, as well as the tasks the visualization aims to support.

Studies have shown that position and length are the easiest and fastest visual variables to decode (Cleveland & McGill, 1984). In my project, I deal with two main quantitative measures: Number of hits and read time (in seconds). To incorporate position, I can plot these two measures on a Cartesian graph with time on the X axis and the selected measure on the Y axis. If I plot each datum as a dot or circle, I end up with a scatterplot. Though, my goal is show trends of popularity. When we think of trends we usually have a mental map of a connected sequence over time — which is why a line chart and area chart are better candidates as they convey patterns of over time, rather than scattered concentrations of data.

As for comparing the total number of hits across different geographic locations, we need to also incorporate another variable — length. Here, bar charts are great candidates for conveying magnitude and for contrasting magnitudes across different categories, as they utilize both position and length. Usually, bar charts are oriented vertically or horizontally. For our purpose, a horizontal bar chart is more suitable for rendering many labels without overlap (Swires-Hennesy, 2014). Lastly, visualizing the number of

seconds for each observation given a geographic location over a specified time period is best accomplished through the usage of position; for this, I can employ a dot chart, which is a one-dimensional scatterplot.

For distinguishing categories in a line or graph chart, color is an effective visual variable. For this project, news categories are not compared to each other, but articles are. Encoding each article with a unique color may work well for up to five colors; however, studies show that surpassing this limit makes it difficult for humans to differentiate encodes of more than seven to nine colors (Healey, 1996). Thus a different visual variable will need to be used to distinguish the articles from one another. Here, I can resort to the visual variable of position. A stacked area graph is an area graph that employs position by stacking different parts of the whole on top of one another against another dimension — usually time (Byron & Wattenberg, 2008).

Computer based visualizations allow data to be processed in real-time and thus enable user interactivity. Interactivity helps enable multiple views of the same data, using the same visualization. Researchers have broken down interactivity into several categories. For this project, I focus on the following: Overview+Detail and Zooming (Cockburn, Karlson, & Bederson, 2008) — and, sorting, filtering, and details-on-demand (Shneiderman, 1996).

3.2 Measuring Time

Time is an essential component to web analytics, as it gives context to what is being measured. Although graphing change over time was attempted in the early A.D.s, the most well known early examples of time-based visualizations were published by William Playfair (Playfair & Corry, 1801). Playfair was the father of the line graph and bar graph. However, there is not a single model that accommodates all domains and tasks (Frank, 1998). For the proposed visualization, time must be dealt with in the context of an online news domain.

Here, it is important to note the difference between event data and interval data. Time events are occurrences that are represented as slices of time; for example, a plane landing at its destination is an event. Whereas, the process of a plane landing occurs over a time interval of approximately 20 minutes. In this project, an article hit is defined as an event. That said, if I present time on an X axis it is inevitable that I will have units of time where no hits are available. The rendering of this time interval as a discrete event is adequate so long as the measured interval is minute compared to the total time covered by the data (Wills, 2011). For example, a visitor reading an article for two minutes is a small amount compared to say an article's lifetime of three weeks. Thus, plotting the observation as it being a slice of time — rather than an interval of time — is adequate.

In order to have a better understanding of the time intervals of news, I consulted professionals around the news station that I work at for feed-

back. Based on this and my observations of our news website and other similar entities' websites, "news" refers to a recent occurrence, notably something that happened today. "Breaking News" typically refers to an event that happened within the current day and whose story is under development. "Recent News" pertains to stories created within the last three days. Most of the older news items are considered archives.

My goal in this project is to represent the "recent news" category. Thus, the needed time granularities to be conveyed that support the task of the proposed visualization should be in days (Bettini, Jajodia, & Wang, X. S., 2000). At a minimum for drilling down to a day's worth of news, I need to use a discrete time scale (Goralwalla, Özsu, & Szafron, 1998) and thus need to capture the day, hour, and minute of each visit. The output variables connected to time should be the number of hits and the average read time spent on an article.

3.3 Existing web metrics systems

What I built is novel in that there is no current navigable and exploratory tool for site visitors, that visualizes near real-time traffic on news sites. Upon researching online, I came across a few systems that specialize in visualizing web metrics:

FLOW

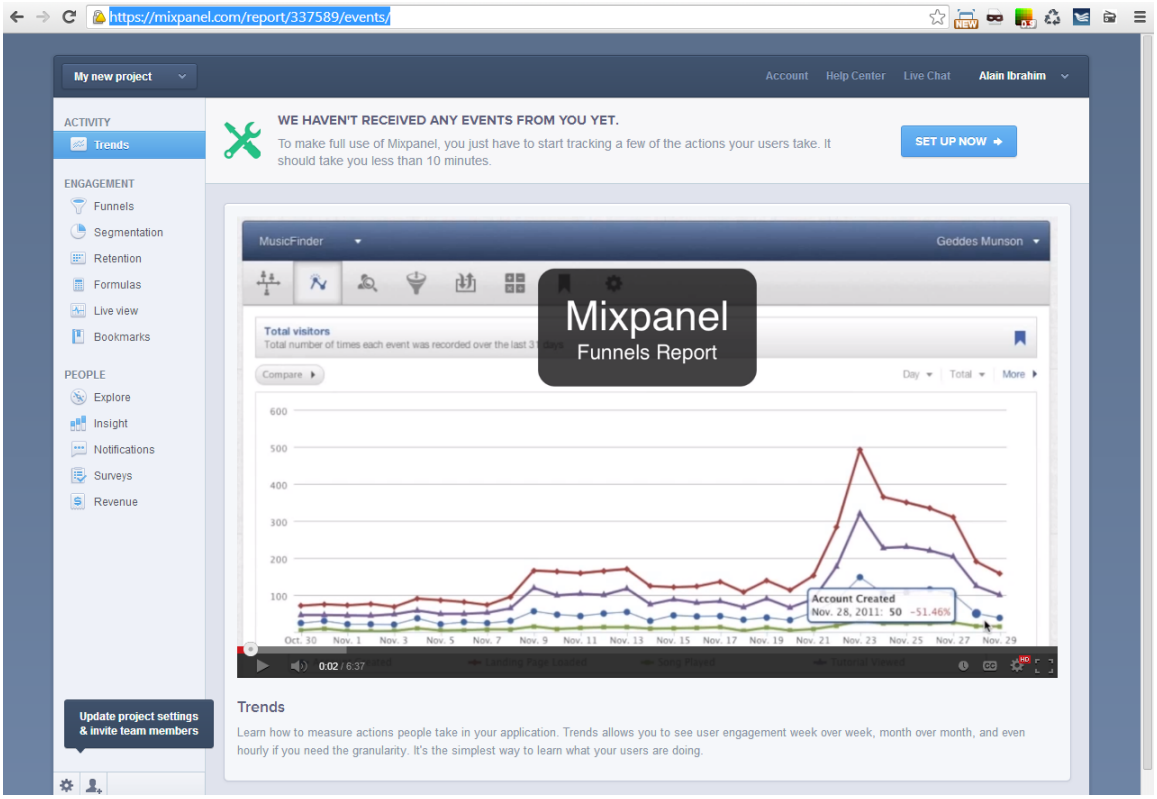


Figure 2 – A line graph in Flow that measures the number of hits on the Y axis, and date on the X axis for select categories on a given website.

While this can be useful in providing some feedback to the news site's administrators, it does not provide detailed insight to the articles' patterns. The below screenshot demonstrates another visualization part of Flow.

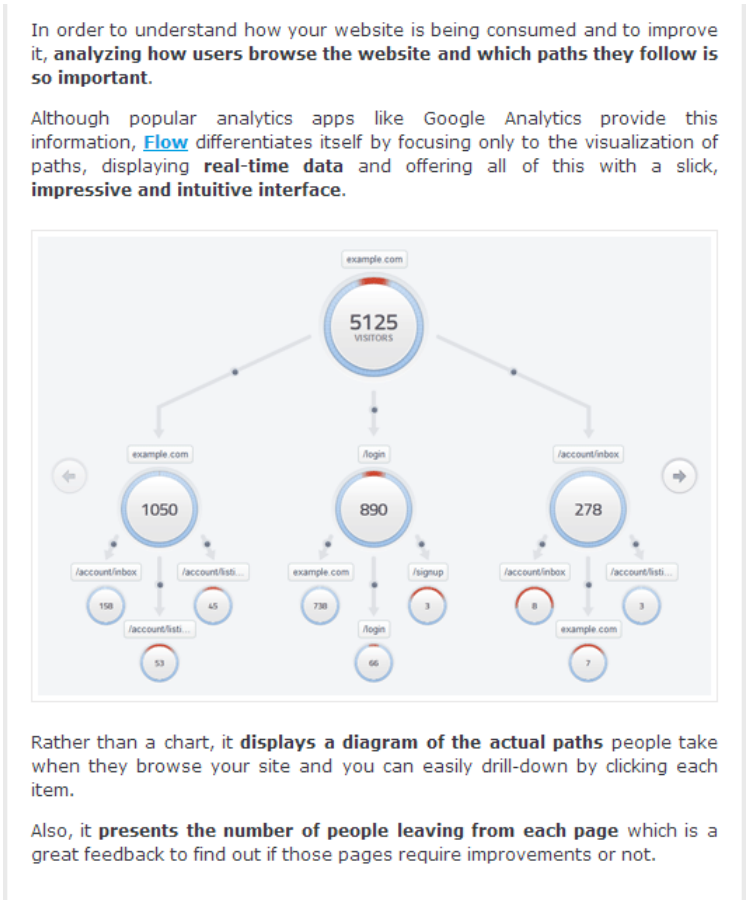


Figure 3 – Highlights of Flow in a hierarchical diagram.

GOOGLE ANALYTICS

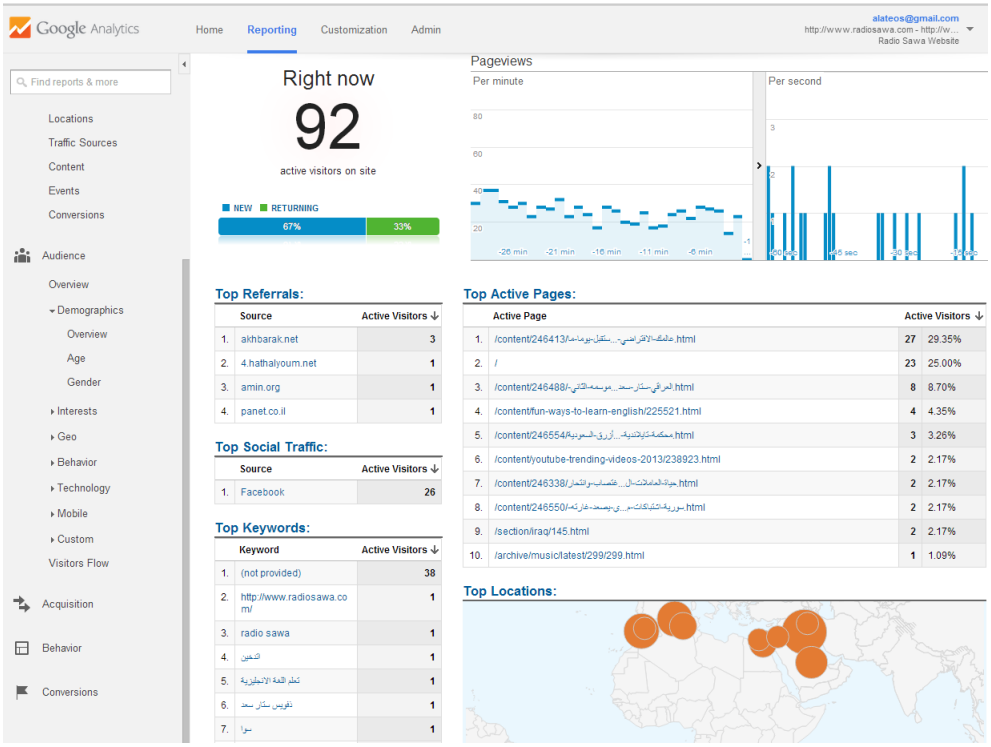


Figure 4 – The Real-Time metrics pane in Google Analytics.

Google Analytics comes in handy in aggregating data and conveying statistical data to a site admin. The admin here navigates with the intention of running an analysis. For example, from what parts of the world is most of this website's traffic coming from? What links are being most visited? While this is a great tool for decision making, it is exclusive to site admins. In addition, it is not user-friendly for navigation.

ADOBE ANALYTICS

Adobe Analytics is a set of tools that cover real-time traffic as well as predictive data analysis based on historical data. It consists of five components: Marketing Reports and Analytics, Ad hoc Analysis, Data workbench, Predictive analysis, and Real-time web analytics. As with Flow and Google Analytics, it contains custom reports and visualizations that are accessed through an interactive dashboard. Once again, the intended audience is that of management and decision makers — not website visitors.

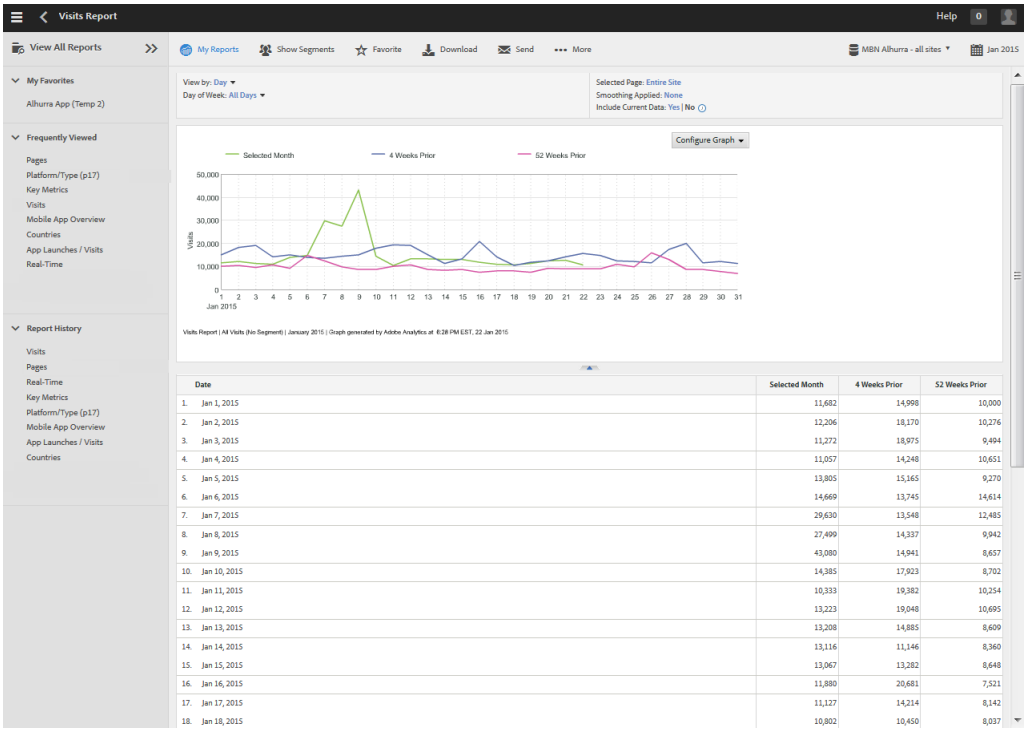


Figure 5 – Hits shown over time in a line graph using Adobe Analytics.

4 Implementation

4.1 Capturing Data

The first step in implementing this project was to determine the necessary web traffic data and where to store it. Some of the data, such as the users' IP addresses and number of hits were captured directly. Whereas, mapping these IP addresses to their corresponding geographic locations required a third party API. Capturing the hits was achieved through server-side script and a relational database. Updating the read time, however, required a combination of client-side and server-side scripting.

4.1.1 IP Addresses

An IP address is a number that uniquely identifies any device that is connected to a network that uses the Internet Protocol. It is the means that I used to track a user. The IP address that I was interested in capturing is the public IP address of the visitor's ISP (Internet Service Provider) router. An IP address can be static or dynamic. The former is set by the network administrator and remains in place unless it is changed manually. The latter is assigned by the network router, and is assigned for a temporary period called the DHCP lease time. After the period of the lease time has expired, the device is given a new IP address by the router. For this reason, I needed to ensure that this IP address did not renew so much as to violate the constraint set by the revisit threshold. The revisit threshold is

a measurement setting which I created to determine whether enough time has passed for the next article hit to be counted as a new one. For example, if the revisit threshold is set to five minutes, and someone's IP address changes every one minute, then the user behind the IP address can visit an article several times within the five minute period and cause the metrics system to register five different hits — each from a different IP address — when it should only register one hit. Thus for the revisit threshold to work effectively, the IP address of the visitor must remain constant from the time stamp of the article hit, and until the time period specified by the revisit threshold has passed.

In order to obtain the average DHCP lease time set by routers, I resorted to a few network engineers and contacted three prominent ISPs (Internet Service Providers). The average DHCP lease time turned out between three and seven days. The question then became, when should the next article hit be counted as a new one? People can visit the same article several times a day. Thus, this threshold is set so as not to over-count the visits in case someone tries to inflate the hits numbers by constantly refreshing the visited article. From a practical standpoint, I see this limit set to a number between five minutes and one hour. Naturally, this figure is arbitrary yet I believe it should not span for many hours. For this reason, it is safe to treat static and dynamic IP addresses the same — as the time revisit threshold would be less than the average renew period of at least one day. This is important as I have defined a hit as a page view that is coming from a unique IP address.

There is a drawback using this method in that it categorizes all users behind an ISP router as the same one user. For example, if me and three people visited the same article in a period of ten minutes, and if the revisit threshold was set to ten minutes, then the code would count all of our visits as one hit. A more accurate way of counting each user would be to use a tracker on the client side, such as a cookie or HTML5's **LocalStorage**. One can completely evade the system by respectively clearing the cookies and by disabling JavaScript, however.

Capturing the hits required server-side scripting. Here, I used PHP. I stored the IP address of the visitor through the use of an associative array called **$_SERVER** that contains several variables pertaining to the HTTP connection between the client and the server. The property that contains the IP address from which the user is viewing a select page is called **REMOTE_ADDR**.

As for user integrity, one cannot fully ensure that a physical human is sitting behind an IP-based device when automated scripts may be hitting the IP address of a given news article. Though, my goal was to minimize the inaccuracy when a visitor reached a news article. I countered this by registering the visiting IP address as soon as the user moved his/her mouse, or, when he/she scrolled at least once when viewing the article at hand. My assumption was that moving a mouse or scrolling a screen indicated the presence of a live being.

4.1.2 Geographic locations

Another metric that I captured was the user geographic location distribution for a select news article. For this, I used the visitor's IP address to resolve his/her geographic locations.

Here, it is important to note that many users may be sitting behind HTTP proxy servers. HTTP proxy servers are physical computers/servers that act as proxy points for the client computers. For example, if someone in Russia used a proxy server located in the US to visit my news article, the requesting IP address that is captured would be that of the US — making it appear that the user visited the article from inside the US. In PHP's **$_SERVER** array, the **REMOTE_ADDR** property contains the IP address of the visitor. Conveniently, there is another property called **HTTP_X_FORWARDED_FOR** which captures the IP address of the visitor while behind a proxy server. So for example, if I were to be sitting behind an HTTP proxy server which had an address of 1.1.1.1, then REMOTE_ADDR would carry that value.
In this case, HTTP_X_FORWARDED_FOR would then contain the value of my router's outside IP address, which is the value I am looking to map geographically. If on the other hand the user was browsing the Internet without the use of a proxy server, then REMOTE_ADDR would carry the value of the user's router's IP address — reflective of the user's geography; and, HTTP_X_FORWARDED_FOR would carry no value.

Here is the method that I used to capture the needed IP address for the geographic lookup:

1. Get the value of REMOTE_ADDR and look up its value with a geolocation API. Call this **X**.

2. Get the value of HTTP_X_FORWARDED_FOR and look up its value using a third party geolocation API. Call this value **Y**.

3. If X and Y are both available, it means that the user is sitting behind a proxy and that the needed IP address is that of Y. If X is available but Y is not, it means that Y is a private IP address of the device behind its router — and thus the user is not behind a proxy server. In this case, X would hold the IP address of the origin.

The next step here was to find a service that would map a given IP address to its corresponding geo coordinates, and ideally to the originating region's name. A quick Google search revealed a convenient REST API service that outputs the responses in JSON. **Telize (www.telize.com)** is a REST API that allows developers to get the visitor IP address and to query the location information for that IP address. Such information includes the country of origin, region of origin, and timezone of origin. The upside of this service is that at as of this writing it contains to no rate limits; thus, one could make an indefinite amount of calls to the Telize API.

For testing, I created a mock web page and applied my code to it. I then visited that page from Alexandria, Virginia. When I did so without using

a proxy server, the hit registered as coming from the United States. Then, when I set my Internet browser to use a proxy server based out of Brazil, the hit still registered as coming from the United States — as intended.

4.1.3 Average read time

Another metric that I was attempting to capture was how long the average user spent reading a certain news article. At the moment, a related measure called the bounce rate checks to see whether the user continues to browse other pages in the visited site, or whether the user "bounces" off to other websites. Meaning, if I visited a certain website, I am tracked as to whether I browse several pages on that website or just bounce (i.e. leave) immediately. Leaving immediately results in a high bounce rate while visiting many pages on the visited site before leaving to another site results in a low bounce rate. Although this measure reflects the interest of the visitor has in the visited site overall, it does not indicate how much time the visitor invested in reading a certain article.

I felt the amount of time spent on a news article indicated how interested the visitor was. However, different articles have different lengths and would result in different reading times despite the level of interest of the reader. For this, I calculated the number of words contained in each article. I then researched the average reading speed of an adult English reader. Based on this reading speed, I calculated the expected time it would take the visitor to finish reading the article. Lastly, I obtained the actual time spent by the visitor to read the article and calculated its ratio

to the expected full read time. I called this ratio the attention span.

Based on an ophthalmology study that consisted of 50 native English-speaking individuals (average age 30.38$\pm$9.44 years; 16 university students, 18 academics, and 16 non-academics) — the average reading speed came out to be 201.53$\pm$35.88 words per minute(wpm) for short sentences, and 215.01$\pm$30.37 wpm for long paragraphs (Radner & Diendorfer, 2014). For the scope of this project, I used the average of 200 wpm to calculate the maximum average time spent on a news article.

4.1.4 Data storage

Based on the data that needed to be captured, the below fields were created:

- **Visitor IP address**: stores the IP address of the visitor reading the news article. In order to avoid capturing page refreshes as new sessions, I set a threshold of ten minutes, whereby the server checks if the visitor has visited the article in the past ten minutes.
- **Time Visited**: stores a time stamp (in GMT) of when the user visited an article.
- **Timezone**: stores the timezone of the visitor's geographic location. This is helpful in determining time differences from GMT.
- **Country**: stores the name of the country where the visitor is accessing the website.
- **Region**: stores the region pertinent to the area within the country where the visitor is accessing the website.
- **Read Time**: stores the time taken for the reader to have finished reading the article. This was calculated in seconds.
- **Expected Read Time**: stores the maximum time it should have taken the reader to finish the words in the body of the article.

- **Article ID**: stores the unique identifier and is captured from the host's WCMS (Web Content Management System). This way, future references could be made easily without having to go through mapping algorithms.
- **Article Publish Date**: stores the time stamp of when the article was published.
- **Category ID**: stores the unique identifier of the category, also as it appears in the corresponding WCMS.
- **Article URL**: stores the article URL.
- **Article title**: stores the title of the article as it appears in the WCMS or on the news site.
- **Sample text**: stores an excerpt of the text from the body of article. This was later used to preview articles in the interactive visualization.
- **Sample picture**: stores a URL to the thumbnail picture of the article.

4.1.5 Database Tables

I adhered to the principles of relational database normalization in order to maximize efficiency and eliminate redundancy. For the scope of this project, all the data pertinent to a web hit was stored in a single table:
hit (id,ip, time_visited, *article_id*, timezone, country, region, read_time)
The underline signifies the primary key. The primary key is a set of one or more fields that is enough to determine the rest of the information for any row instance. In this case, the IP address, article ID, and time of visit jointly determine the timezone, country, region, and read time. However, since I needed a way to track the user's visits to capture read time, I designated an auto-increment unique identifier called **id**. *article_id* is italicized since it is a foreign key — that is, it represents a primary key in another table. The time visited is rounded to hour:minute. Therefore, seconds are not be accounted for in the hit's time stamp. The read time defaults to one second, which signifies the shortest time a user would spend when bouncing away to another site. In the event the user spends more time reading the article, then the actual time spent is captured here.
The second table contained all the metadata pertinent to the visited article: **article (article_id, publish_date, category_name, article_url, title, sample_text, sample_pic, expected_read_time)**

The third table contained metadata pertinent to the category of the visited article: **category (category_id, category_name**)
Here, it is important to note that all of above data is communicated via the server side script which is included in the code of each loaded article.

For this project, I used MySQL since I have good experience with it and for its being open source. MySQL is a relational database management system. Relational databases are those that adhere to relational database design, which relies on the fundamentals of relational algebra and relational calculus. The main tenet of relational design is to maximize efficiency and eliminate any redundancy in the data being stored (Codd, 1970).

4.1.6 PHP Server-Side Code

Retrieving the web metrics involved creating configuration files, classes that abstracted the database tables, and code that performed the queries and outputted the resulting data in JSON.

Each table in my database was represented by a PHP class. Hence, the creation of the following three classes:

1. **Hit.php:** This class registers all of the metadata pertinent to a user visit. This data includes IP address, country, region, timezone, and time visited.

2. **Article.php:** This class captures all of the metadata pertinent to the article being visited. Such includes title, text excerpt, thumbnail URL, and article URL.

3. **Category.php:** This class captures any metadata pertinent to the parent category of the visited article. At the moment, I am only capturing the ID and name of each category.

In addition to the above, I created a helper file that registers the read time of a given article. By default, the read time is registered as one second in the database. Once a hit is registered in the database, the ID of the newly created hit is then passed to the DOM of the article page. There, the ID is used via an AJAX call to store the calculated read time for the visited article in the database.

4.1.7 Client-Side Read Time Capture with JavaScript

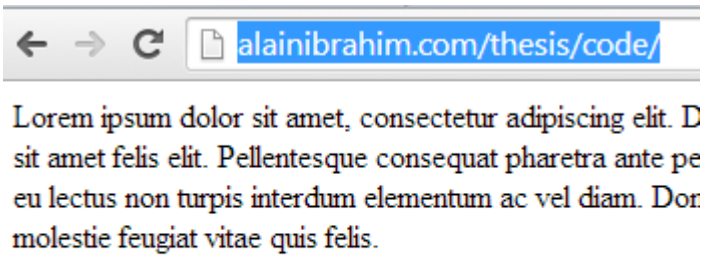
As mentioned in the last section, the hit ID is passed to the client page in order to keep a way to identify and update the read time for the visited article. Here, I calculated the number of words that are stored in the article. I did this by first getting all of the text found within common article tags — namely , , <article>, <p>, and <section>. I then summed the number of words found within all of the mentioned tags. I ensured that double counting was avoided by using the **innerText** property; this counted the elements' inner text, but not any text that might have been included in one of its children nodes. Next, I calculated the number of seconds the article was on average expected to take to read. This was based on the human reading speed of 200 words per minute figure that I obtained earlier. This figure was used in the event that a user had left the page open indefinitely. Thus, if an article was expected to take about five minutes to read, and the user took (for whatever reason) 30 minutes to navigate away from or to close the article, then I registered the five minutes. Although I may have compromised some accuracy here, it remained much more advantageous than having significantly inflated the captured read time.

Once the article page loaded, and the expected read time calculations were made, a timer was readied. The timer was triggered once a user moved the mouse or when the user scrolled the page. This minimized storing metadata created by automated scripts and to a high extent ensured

that a human was sitting behind the screen. Upon navigating away from or closing the article, the read time was then submitted to **updateReadTime.php** and was eventually recorded in the database. This part of the implementation relied on the JavaScript events **window.onbeforeunload** or **window.onunload**, depending on the browser's user agent. At the time of this writing, this method was tested successfully with Google Chrome 39.0.2171.99 m, Mozilla Firefox 34.0.5, Safari 5.1.7, and Internet Explorer 10.

4.1.8 Testing the Capture of Metrics

Now that both the back end and front end code has been implemented, I was able to move on to testing this system. I created a miniature Lorem Ipsum page which acts as an article being visited:



In my PHP code, I populated mock metadata for the article and category. The hit metadata, however, was populated based on the visit data. I went ahead and visited the page for ten seconds. My ip address, time of visit, fake article ID, timezone, country, region, and read time were all stored in the **hit** table. The article ID, category ID, article URL, article title, article sample text, and the URL to the article's main pic were all stored in the **article** table. Lastly, the category ID and name were both stored in the **category** table.

This concludes the part of the system that implements the capture of the web metrics. The next step was to connect this library to a working website. For the scope of this thesis, I implemented the hook to specifically tie into a widely used Web Content Management System — Wordpress.

4.2 Hooking into Wordpress

Wordpress is a web-based Web Content Management System (WCMS). It is essentially a web application that allows its users to create and administer posts, which are ultimately showcased on a website to site visitors. In this project, each news article is implemented as a post and can be categorized under one or more parent categories. Wordpress is very common for its being free and open source; hence, it was a great candidate for me to test my system against.

For each post there is a template file called **single.php**. This file determines how the article is displayed and what content will be outputted to the visitor. It can be found under the current theme that is being used. A theme in Wordpress is a set of pre-built PHP, HTML, CSS3, and JavaScript templates that structurally and visually dictate how the resultant website looks and behaves. In order to accomplish the task of capturing the data from a certain article, I needed to include my capture library inside **single.php**. However, since my library cannot directly communicate with Wordpress, I had to create a hook that abstracts some of Wordpress's built-in functions. Lastly, I needed to ensure that Wordpress was using jQuery version 2.1.1 or higher. jQuery is a JavaScript library that shorthands many of the functions needed to interact with the web article's HTML document. Since I needed this library to load before the article page fully loaded, I included jQuery in **header.php**, which in Wordpress comprises the <head></head> portion of all pages in a given website.

4.2.1 Creating the captureLib Folder

In the interest of portability, I have placed all of my code within a folder called **captureLib**. If a third party wishes to uses this library in the future, all that would be needed is to simply include **captureLib** from within **single.php**.

Below is a breakdown of the current directory structure of the code that captures hits and read times.

1. **lib/**
   (a) **constants.php**: contains all the server-side constants that will be used to capture the hits.
   (b) **db_connect.php**: connects to the database where the hits are stored.
2. **Article.php, Hit.php, Category.php**: as mentioned previously, these are abstractions of the database tables that communicate the data from and to the database.
3. **getReadTimeLibrary.php**: a line of code to include the read time JavaScript code if a hit was registered.
4. **readTime.php**: the JavaScript code that captures the read time. It is terminated with .php (vs. .js) since it requires PHP code to output the hit ID inside the JavaScript file. I have made adjustments to this code

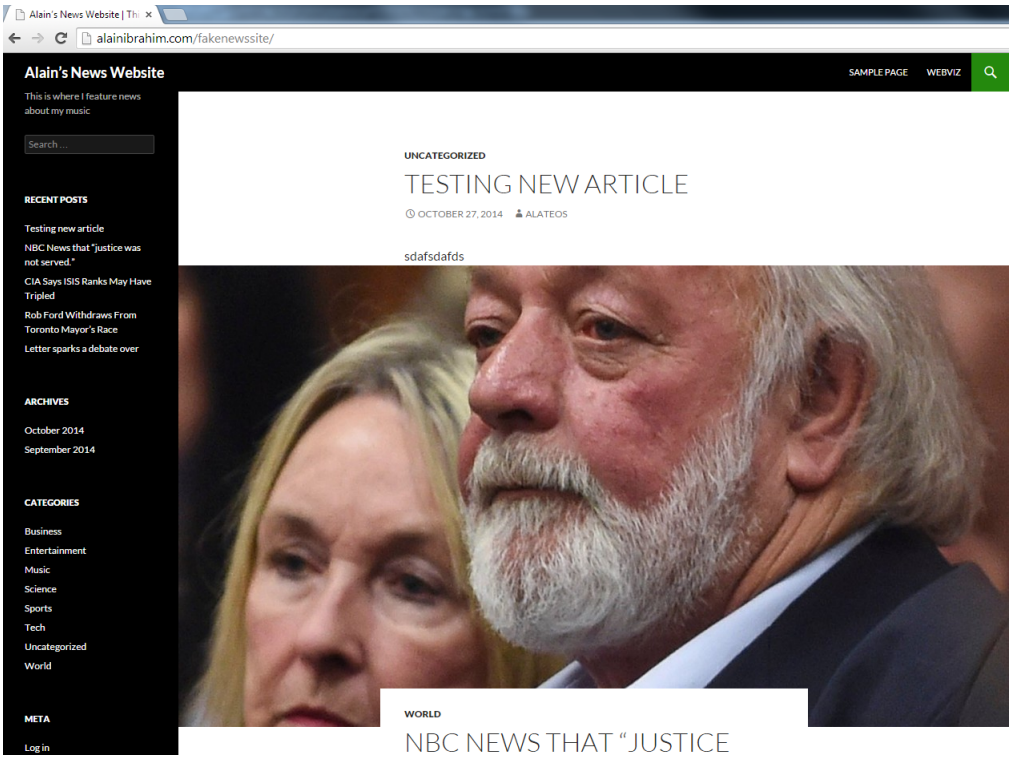
to accommodate for users switching tabs while reading a certain article. When the article's window loses focus, the timer stops counting; it only resumes when the user gets back to the article window. This maximizes accuracy in the average read time readings.

5. **wordpressHook.php**: as mentioned earlier, this is an abstraction into Wordpress's needed built-in functions. In order to hook to different WCMSs, the code can be modified and reused.

6. **countries.php**: contains an array that has all country names. Used for database seeding.

4.2.2 Creating a Mock News Site

In order to test the capture library, I created a mock news site in Wordpress. In this setup, I built the following categories: Business, Entertainment, Music, Science, Sports, Technology, and World. I tasked a friend to populate mock articles with content that is fitting to the parent category. I created the Wordpress install under my personal web hosting account. The URL to it is http://www.alainibrahim.com/fakenewssite. In the interest of time, I used the default theme that came out of the box.



4.2.3 Testing

I visited four different articles, spent different amounts of time on each, and navigated directly from one article to the next. The categories metadata of the different articles that I visited were stored in the database. For this project, I only stored the categories' ID and name, as they appear in the Wordpress database tables. Note that if I visited an article of the same category again, the category was not stored twice.
The articles' metadata were successfully stored. The sample text was excerpted by PHP with a default setting of 500 characters. This text was later used for previewing articles. The read times (in seconds) were accurately registered in the **hit** table.

4.2.4 Seeding Mock Data

After testing the capture of data, I needed a way to mimic traffic on a real news website. In addition, I needed to make it appear as if each mock article visit came from a random country of origin. Below are the steps I took to accomplish this:

1. Modified **constants.php** to include a flag that denotes whether the app is in test or production mode:

```
...

// defines whether we are in test or production mode
define(PRODUCTION_ENV, false);
```

2. Modified class **Hit.php** to produce a fake read time and fake US state of origin if in test mode:

```
...
// if in test environment, populate fake data for read time and country of origin
if(!PRODUCTION_ENV) {
        // get a random time between 10 seconds and 3 minutes
        $read_time = rand(10,180);

        // get the list of all the states in the US, and pick one randomly
        include("states.php");
        $region = $states[array_rand($states)];
}
...
```

3. Created **makeFakeHit.php** to invoke a visit to a random article on my fake news site:

```
/**
        The sole purpose of this file is to visit a random article on the
        fake news site, causing hits to register
*/

$TESTING_IP = "98.172.153.142";

// 290 is the id of the last fake article that was created
$article_id = rand(1,290);

// cause a fake visit only if testing ip address is registered here
if( $_SERVER['REMOTE_ADDR'] == $TESTING_IP) {
        file_get_contents("http://alainibrahim.com/fakenewssite/?p=" . $article_id);
}
```

Note that I have hard coded a testing IP address so that I am the only person who can run this for testing.

4. Created **fakeVisit.html** to run on the client side and to call **make-FakeHit.php** at a pre-specified time interval(using JavaScript):

```
(function(){
        // the amount of time to wait before revisiting the site (in seconds)
        var REVISIT_INTERVAL = 20;

        // trigger the fake visit on the server side
        var makeFakeHit = function() {
                $.get("makeFakeHit.php");
        }

        // set a recurrent loop to seed the data
        setInterval(function(){makeFakeHit();console.log("hit issued")},(REVISIT_INTERVAL*1000));
})()
```

4.3 Retrieving Captured Metrics

At this point, the database was populated with a good amount of data and I needed a way to retrieve it. For this, I created a folder on the server side and called it **vizLib**. In this directory I created three PHP files that outputted the results in JSON:

1. **getCategories.php**: outputs all the used article categories.
2. **getCategoriesMetrics.php**: outputs all of the metrics for all categories.
3. **getArticlesMetrics.php**: outputs all of the metrics for all the articles given a category ID, beginning visit time, and ending visit time.
4. **getArticleCountriesMetrics.php**: outputs all of the country metrics for a given article ID.
5. **index.php**: the landing page for the visualization.

4.4 Visualization

This brings us to the final piece of the project — conveying the web traffic to the visitor with an interactive and navigable visualization.
As mentioned earlier, the goal of the visualization was to support the following tasks:

1. To compare articles' hits and reader attention spans for any period within the last month, given a select category.

2. To preview each article's thumbnail, title, and excerpt.

3. To contrast the geographic distribution of hits and attention of up to two articles.

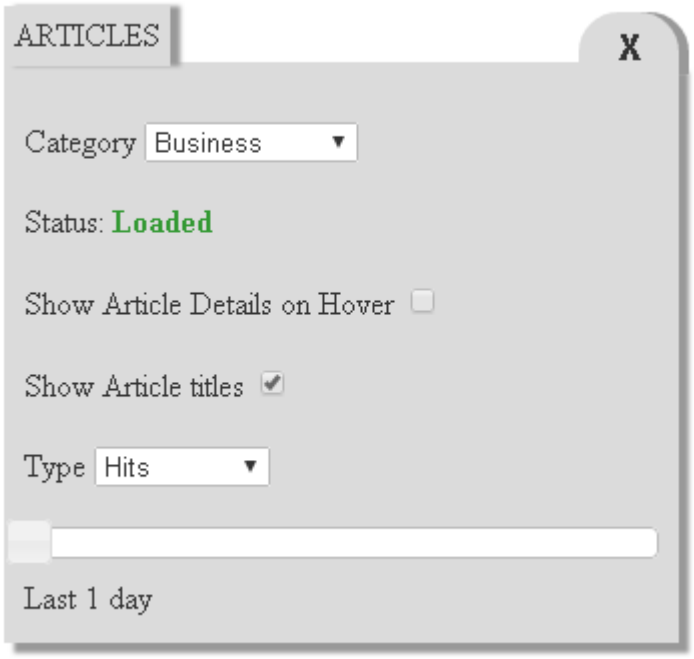
4. To navigate to the full article on the news website.

4.4.1 The controls box

The controls box helped the visitors filter the news categories and time range of the articles hits. Typically, controls take up space in the limited real estate area available in any web browser. Though, I felt it would be best to allocate this space to the visualization. For this reason, I made the controls box drag-able and hide-able.
The controls box first partially loads when no category is selected. The clickable "X" hides it; when hidden, the top left gray rectangle with the two white dots is contoured by a black border.

Upon loading the visualization, an AJAX call is made to the server which fetches all the of the news categories into this dropdown box. Once a category is selected, a call is made to the server, which results in all of the data for the selected category to be fetched from the database using the server-side script.

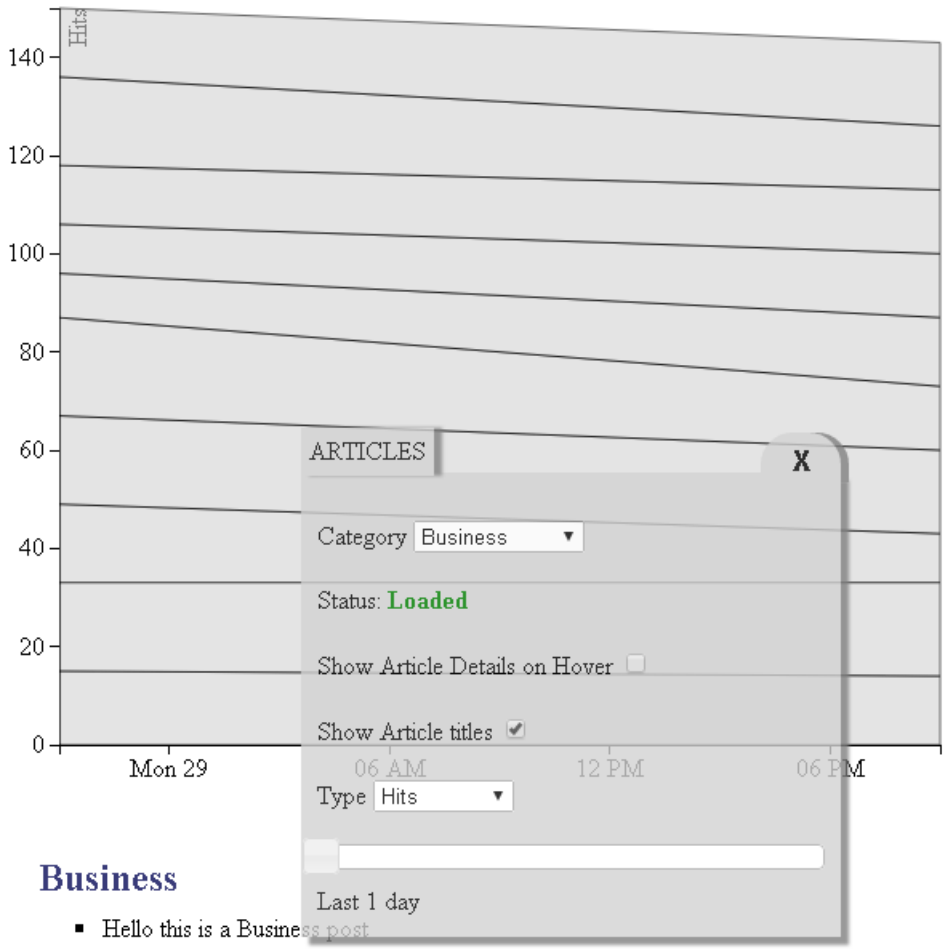
Once the data is loaded, the status label turns green and reads "Loaded":



Below are the descriptions for each component of the controls box:

- **Category:** Sets the category whose articles the visitors choose to view.
- **Status:** Shows the data transfer status of the selected news category.
- **Show Article Details on Hover:** Determines whether a preview of the article will pop up upon hovering over an article in the stacked area chart.
- **Show Article titles:** Toggles the visibility of the article titles in the bottom portion of the screen.
- **Type:** Toggles between viewing the number of hits and amount of time spent (in seconds) on the Y axis of the stacked area chart.
- **Time slider:** Allows the user to select a day or range of days to filter the hits.

I made the controls box transparent in case the user wants to keep it afloat the visualization:



Business

- Hello this is a Business post
- Dollars VS People?
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- $3 Per Gallon Gas, Here We Come
- Are you putting food on your table?
- Madoff Son Andrew Wills More Than $15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Facebook Tests Disappearing Posts Feature
- August PS4 and Xbox One Sales Continue to Break Records

4.4.2 The Stacked Area Chart

Here, the goal was to create a visual product that would allow the user to view all the articles' popularity over time, given a certain news category. I knew that I wanted time to go on the horizontal X axis, and hits or read time to go on the Y axis. Based on this, I first started with a simple line graph, where each line represented an article and was distinguished by its color. There were two downsides which led me to abandon the line graph:

- The lines intersected into a spaghetti mash-up, making it very difficult to spot any significant patterns.
- When a category (e.g. Business) contained more than eight articles, I had to use eight different colors to encode the articles. The issue here was that at a certain point, colors became very similar making it almost impossible to visually follow the article all the way through a selected time period.

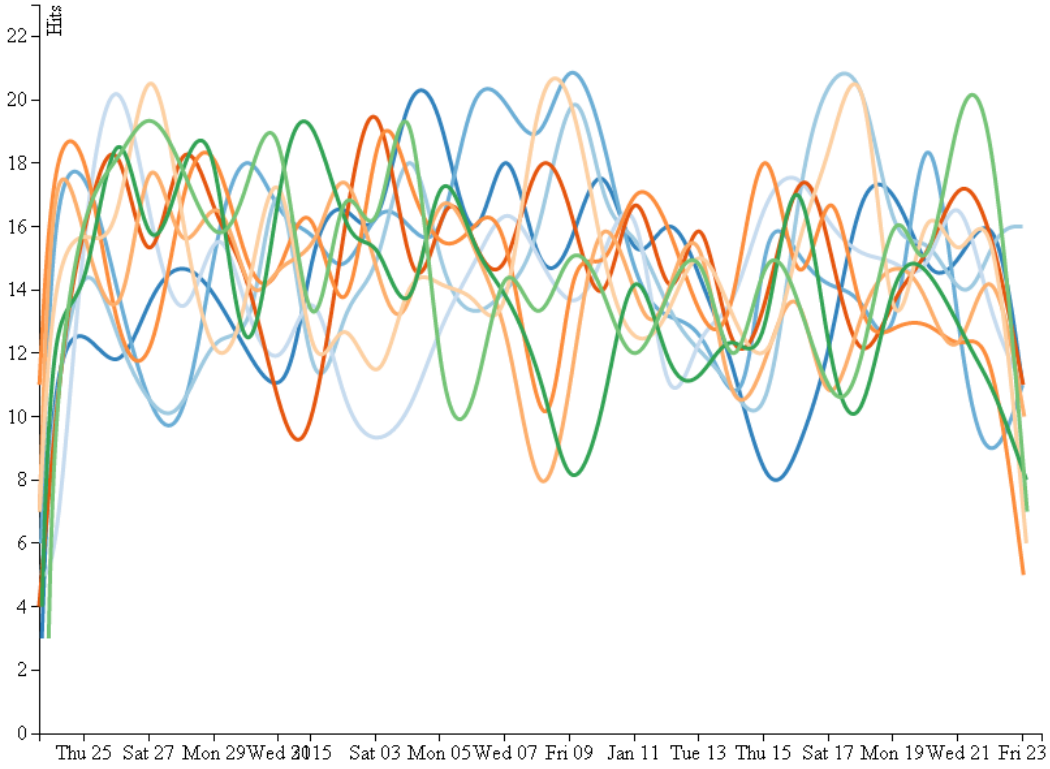
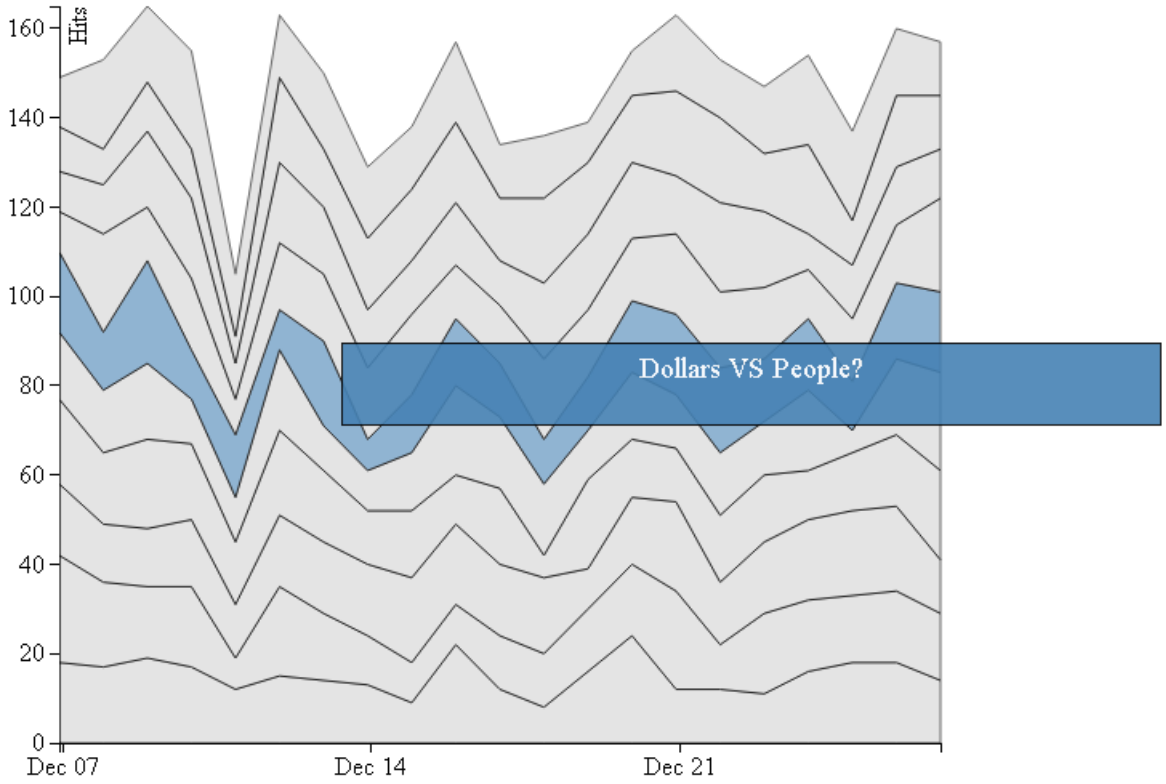


Figure 6 – Article web metrics shown in a line graph. Each article is represented by a line and is distinguished by its color.

I instead used the stacked area chart. The stacked area chart maintained a view on the patterns without compromising legibility. It mainly supported the task of comparing articles' hits and reader attention spans for any period within the last month, given a select category. It also supported the task of previewing an article's thumbnail, title, and excerpt.

The X axis is laid out using D3 and represents time aggregated on an hourly basis:

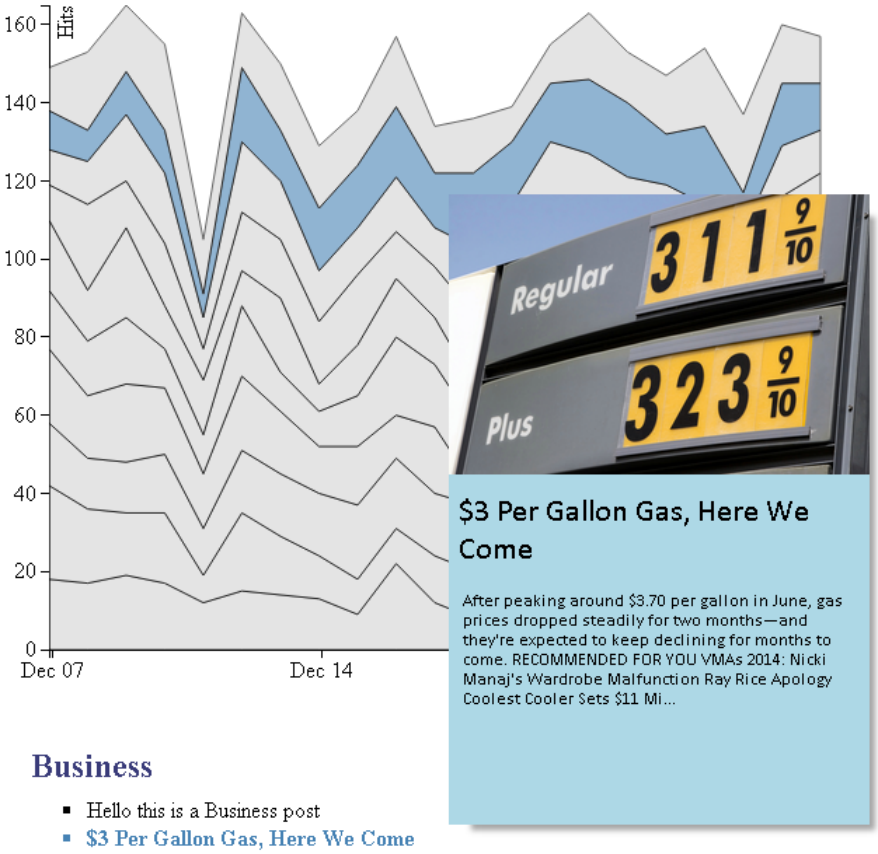


Business

- Hello this is a Business post
- $3 Per Gallon Gas, Here We Come
- August PS4 and Xbox One Sales Continue to Break Records
- Facebook Tests Disappearing Posts Feature
- **Dollars VS People?**
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Madoff Son Andrew Wills More Than $15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Are you putting food on your table?

The articles are stacked vertically, each respectively matching its title in the below white space area, which is headed by the selected category. The readying of the data for the stack is done using the D3 area layout, and the visual representation is implemented by using the SVG <path>. The paths by default all contain a fill of light gray.

Hovering over any article in the stack chart causes both the article's area and corresponding title to turn to steel blue. In reverse, hovering over an article title in the bottom area causes the corresponding article's area in the stack chart to have a fill color of steel blue. It is important to note that the user has the option to see a preview of the selected article when checking **Show Article Details On Hover** in the controls box:

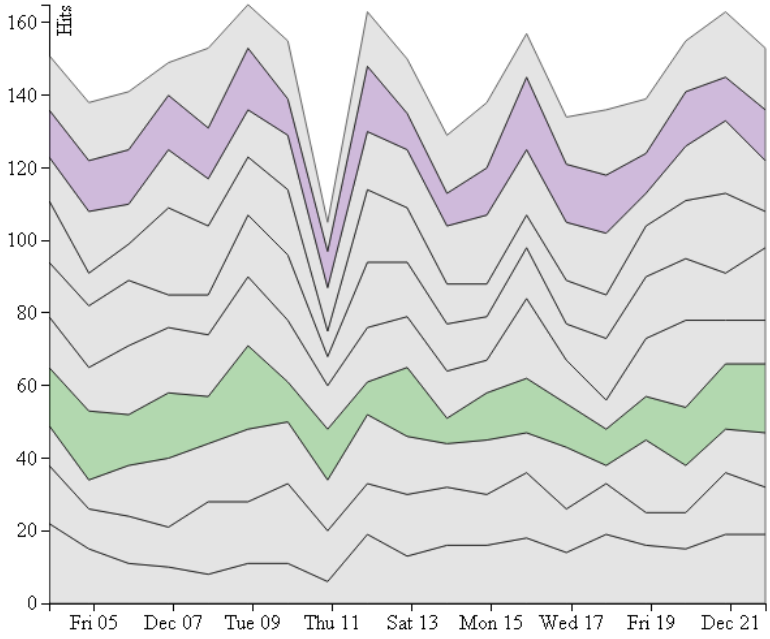


Business

- Hello this is a Business post
- **$3 Per Gallon Gas, Here We Come**
- August PS4 and Xbox One Sales Continue to Break Records
- Facebook Tests Disappearing Posts Feature
- Dollars VS People?
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Madoff Son Andrew Wills More Than $15 Million to His Fiancée and Family
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- Are you putting food on your table?

This results in an overlay to pop over the selected article that includes the article's thumbnail, title, and a 500-character excerpt of the body of the article. The user is given the option to contrast the pattern of up two articles over time, by left clicking on the select article area. This results in one article being filled with a purple color and the other with a green

color. I picked the colors from **colorbrewer2.org** and ensured they were colorblind safe.



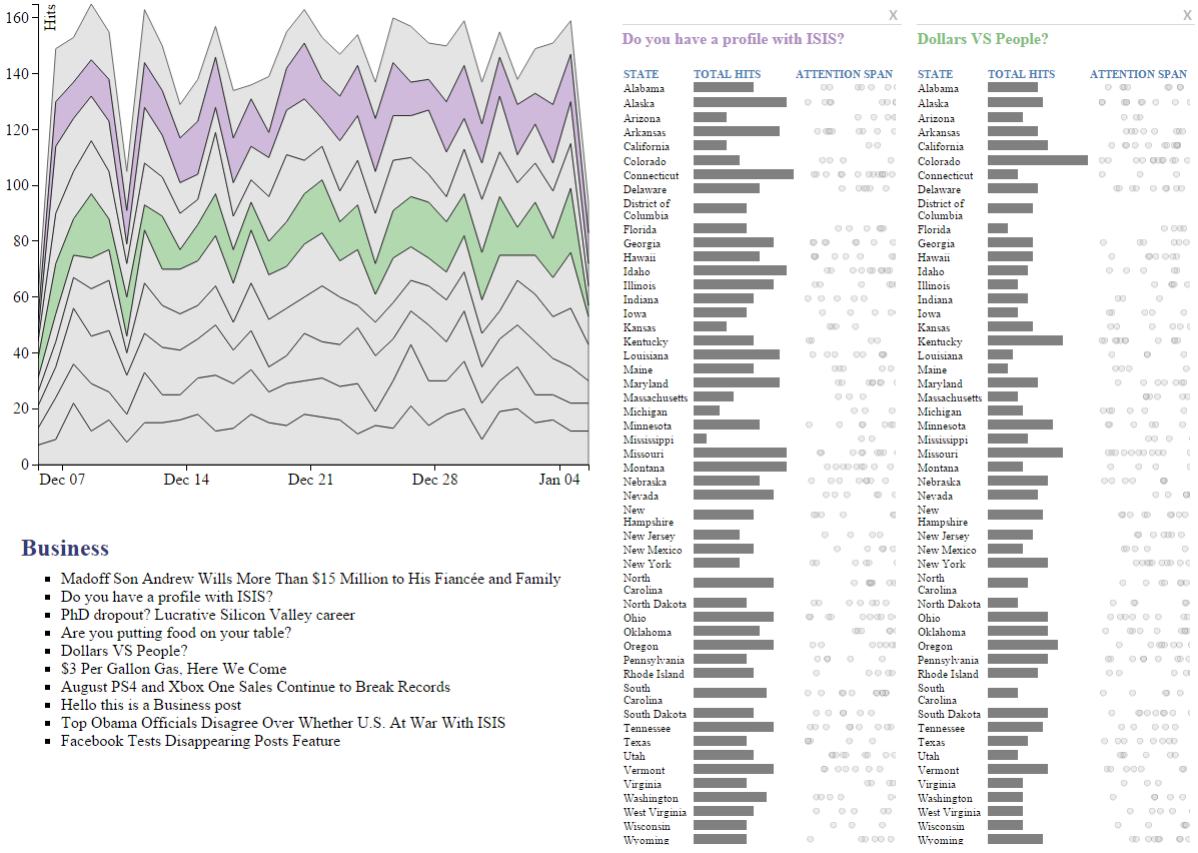
Business

- Facebook Tests Disappearing Posts Feature
- Top Obama Officials Disagree Over Whether U.S. At War With ISIS
- Do you have a profile with ISIS?
- PhD dropout? Lucrative Silicon Valley career
- August PS4 and Xbox One Sales Continue to Break Records
- Are you putting food on your table?
- Dollars VS People?
- Madoff Son Andrew Wills More Than $15 Million to His Fiancée and Family
- Hello this is a Business post
- $3 Per Gallon Gas, Here We Come

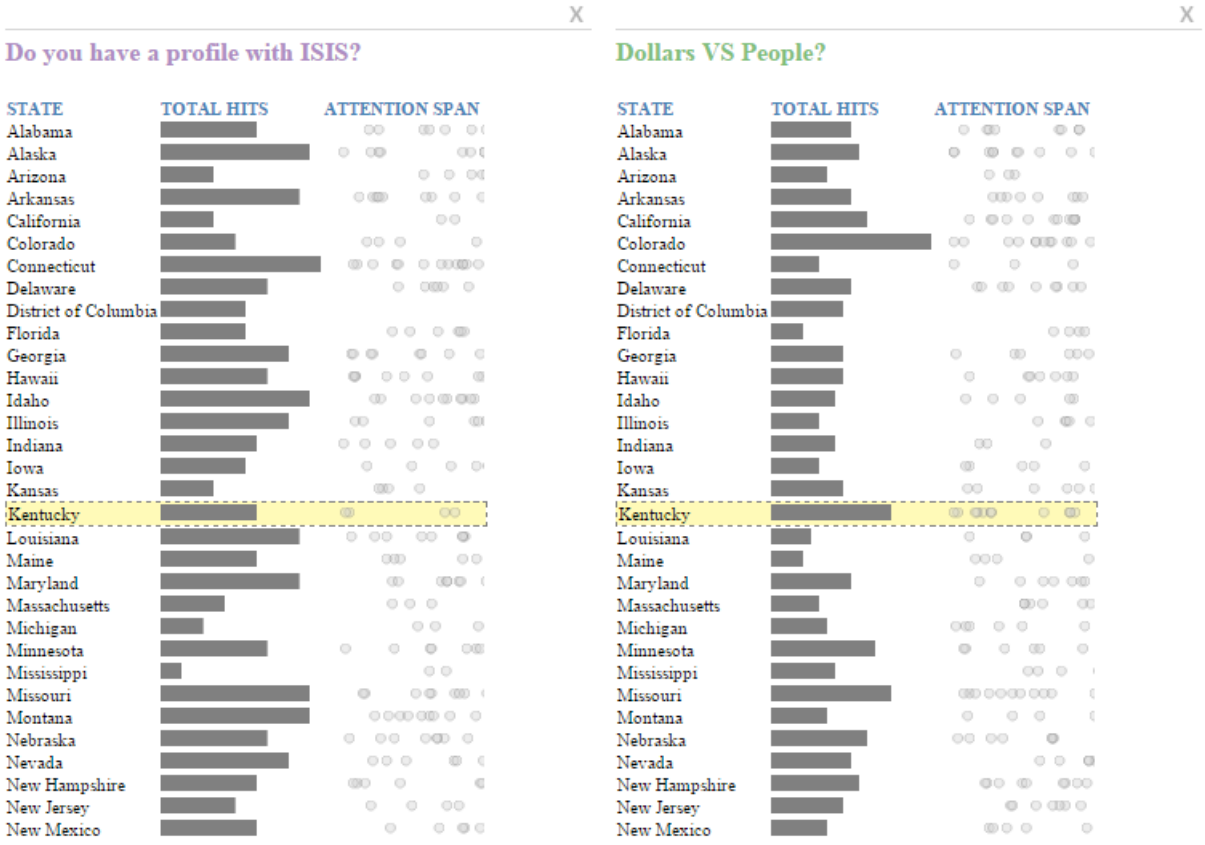
Once the two articles are selected, their corresponding geographic distributions are shown in the right area of the browser. This is covered in the next section.

4.4.3 Geographic bar charts

In the previous section, two articles are clicked on in the stacked area chart. Now, in addition to their being highlighted in the stacked area chart, a right side pane populates each article's title in its color, along with a table below it. The table contains the geographic origin (e.g. state/country), the total number of hits, and the attention span distribution (i.e. read time in seconds).

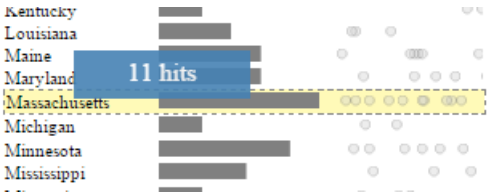


Whenever a first article is selected, the second article is compared along the same geographic areas for uniformity. That is, only the geographic areas that are included in the first article are displayed for the second article. Clicking the top right "X" removes the article from the right pane, and renders the graph area counterpart back to light gray. The hits values are encoded as horizontal bars in dark gray. The attention span column contains a distribution of the captured read times, represented as light gray circles; the amount of seconds spent is encoded by the horizontal position of each circle. The circles are transparent to allow for overlap. The maximum horizontal position of the circle is demarcated by the expected read time. The expected number of seconds was discussed earlier, and is a measure that takes into account the average adult human reading speed and the number of words contained within the article. The total hits bar spans relative to the highest hits value for that article — meaning, the bar that takes up the biggest width is the highest hit value that was registered for that article.



We now examine the geographic area in more detail. Note that automatically, the entries are sorted by the name of the geographic origin, in ascending order. Clicking on the **STATE** header toggles back and forth the sorting order of the geographic origin name. Similarly, toggling the **TOTAL HITS** header sorts the data by number of hits in ascending/descending order. When the data is sorted, it is sorted according to the same criterion for both articles.

Hovering over a row causes that row to be highlighted in yellow, and to be contoured in a dashed line solid border. Hovering over any hits bar inside a row pops up a steel blue overlay which displays the number of hits:



Similarly, hovering over a circle inside a row pops up a steel blue overlay which displays the read time in seconds:

4.5 Results

4.5.1 Mock Website

The working example that I have discussed so far is based on a mock website that I created using fake visit data. The automated visitor script ran on a daily basis for three months. For each hit, the read time was captured as a random number between ten and 180 seconds. When I switched the flag in my PHP code to denote the production environment, the read time was captured as the actual time I spent on the news article. Most of my testing was done in Google Chrome version 39.

I initially reserved the geographic data for the articles to be serviced by **getArticleStatesMetrics.php**. However, this resulted in more calls to the database server. I felt that in a production environment this would slow down the system and affect all users. Thus, in order to minimize the number of requests to the server, the hits data was retrieved via **getArticlesMetrics.php**. This fetched all data for a given news category — including the geographic origins of the hits — for the last 31 days. The client-side JavaScript code then parsed and re-organized the fetched data to be used by the interactive visualization.

The final product included a stacked area chart, a list of article titles below, and a right pane for geographic analysis of up to two articles. The hits were first aggregated on a minutely basis. This resulted in performance degradation. Adjusting the date range had a latency of about two to three seconds to reflect in the redraw of the stacked area chart. I then

aggregated the hits on an hourly basis, which ended up plotting Y values over 744 (31 days x 24 hours) X axis time values. This eliminated the latency in the stacked area chart redraw.

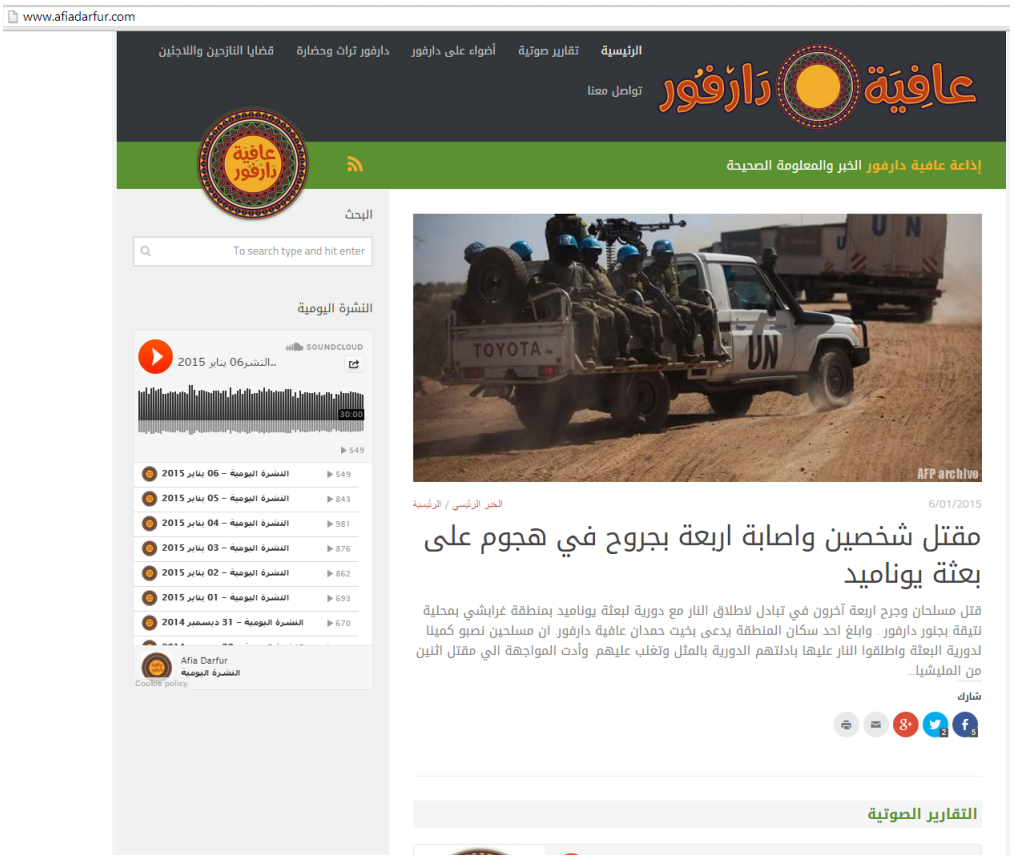
The stacked chart accurately reflected the number of hits and average read time over the selected time range. Clicking one or more articles highlighted the article area in the stacked area chart and displayed the total hits based on geography. It also showed the distribution of the recorded read times. For the mock site, the geography was limited to states within the United States. The tables that contained the geographic origins were sortable by total hits and by state name — in ascending and descending order. When **Show Article Details on Hover** was enabled, the pertinent article's thumbnail image, title, and body excerpt were successfully presented in an overlay. Hovering over any row in the geographic hits and attention span table resulted in the row being highlighted in yellow for all articles that are being compared. Hovering over a bar representation of the total number of hits invoked a steel blue overlay showing the number of hits. Lastly, hovering over an attention span circle invoked a steel blue overlay showing the number of seconds that was spent on the article.

In the mock site, the last 31 days amounted to 4500 hits per category. The time it took for them to be loaded in the visualization was around four seconds, using a PC with an Intel i7 processor and 12 GB of RAM. When changing the date range, the stacked chart's redraw time was almost instantaneous. When **Show Article Details on Hover** was enabled, there was a delay of about 500 milliseconds for the image thumbnails to initially

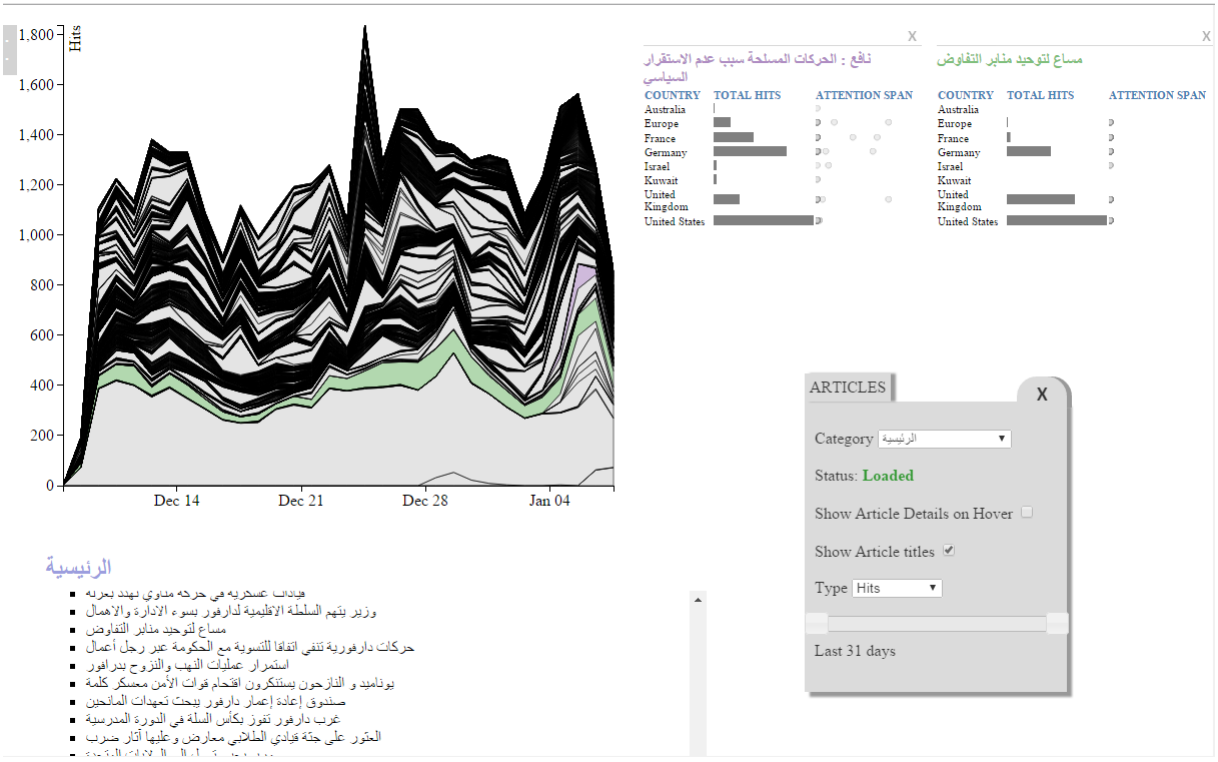
show. Once they became cached in the browser's memory, they appeared without delay. For images that were not cached, the delay was about one second per megabyte in picture size.

4.5.2 Live website

Being a web developer at a news organization, I was able to test the system against one of our mini sites — Afia Darfur (http://www.afiadarfur.com), since it was using Wordpress as its back-end platform. The site targets visitors from Sudan and keeps them up to date with news specific to Sudan.



When I first hooked my code into **afiadarfur.com**, I experienced issues with both the capture and visualization ends of the system. The language was Arabic and thus used a different character set, which caused some code statements to break. In order to overcome this, I changed the collation (i.e. character set) of the database tables to **utf8_general_ci**. The PHP library files responsible for capturing data also needed to be encoded in UTF-8. Once fixed, the capture library started to work. One month later, I had enough data to identify patterns:

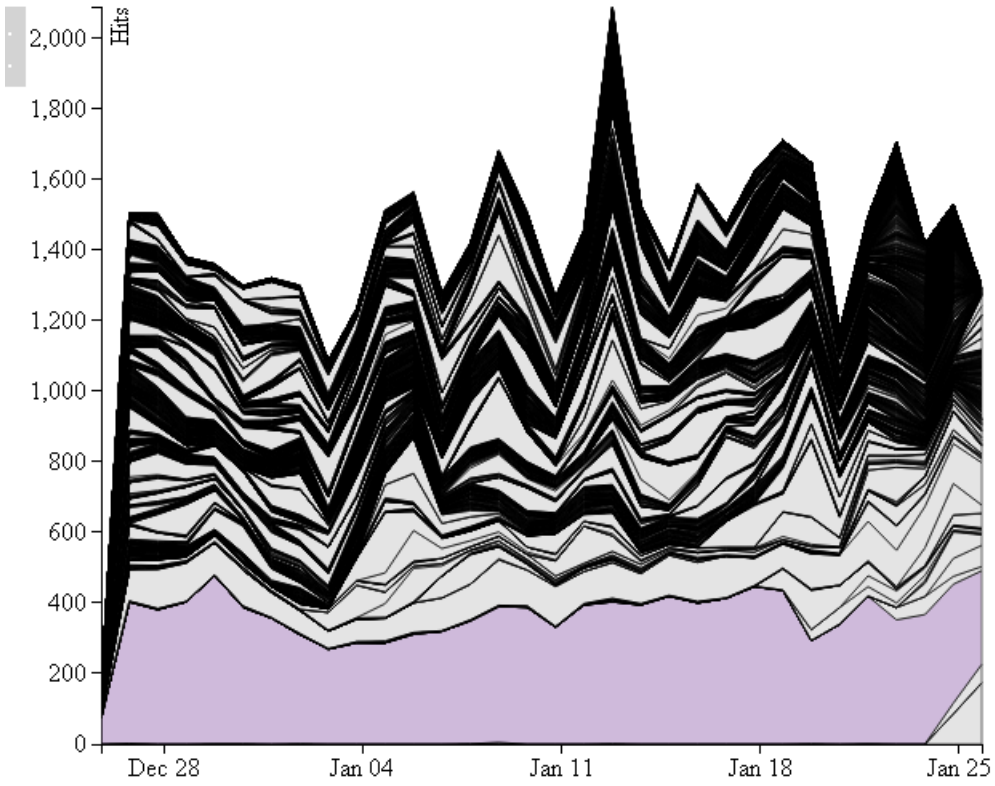


When enabled through the controls box, the article previews were successfully triggered:

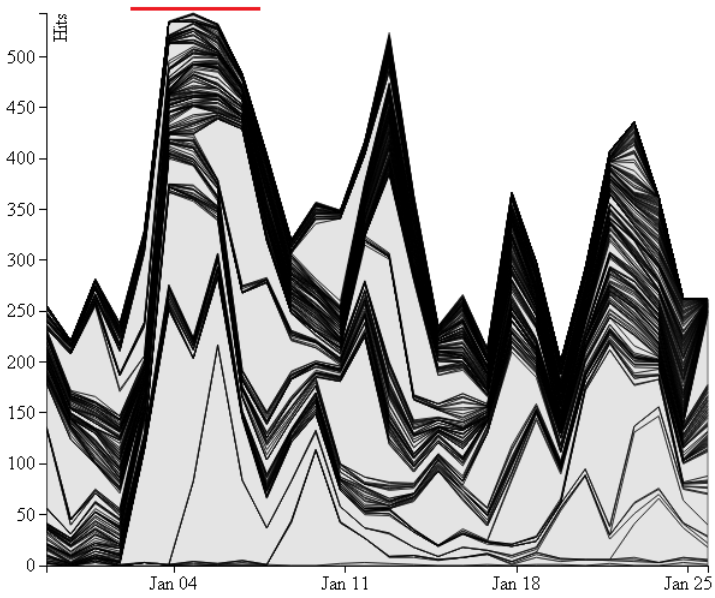
The resultant visualization had the same appearance as that pertaining to the mock news site. The interactive capabilities were identical. The main difference was the type of locale being displayed in the geographic distribution tables. In the mock site, the geographic locale of interest was the **US state**. Since the main audience for Afia Darfur is internationally spread, I designated the locale of interest as **country**.

When I examined the visualization in closer detail, I came across three interesting observations:

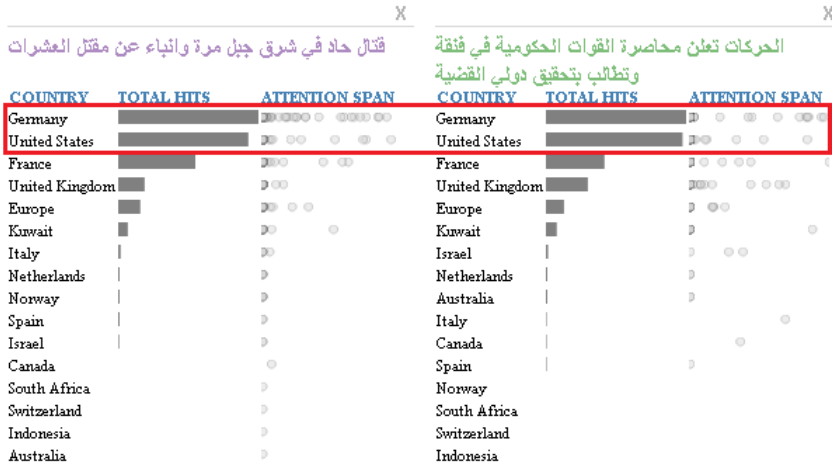
- There was a particular article in the "main news" category which had a considerably higher number of hits when compared to that of other articles. The title of the article translates to "UNAMID implement development projects in Darfur". On average, it dwarfed the other articles by constantly showing around 300 hits. It is highlighted below, in purple:



- For the "leading news" category, there seems to be an abnormal spike of hits for around the period of January 4, 2015:



- Most hits came from Germany and the US. Somehow, Sudan was not even included in the mix:



In the Afia Darfur website, the last 31 days amounted to 10,000 hits and 40,000 hits for the two main categories being used by the web editors. It took approximately 16 seconds to load the 10,000 hits and 52 seconds to load the 40,000 hits. As for adjusting the stacked area graph's time range, there was approximately a three-second delay for the 10,000 hits and approximately a ten-second delay for the 40,000 hits. This was done using a PC with an Intel i7 processor and 12 GB of RAM. When **Show Article Details on Hover was enabled**, there was a delay between 50 to 300 milliseconds for the image thumbnails to initially show. Once they were cached in the browser's memory, the images appeared without delay. For images that were not cached, the delay was about one second per megabyte in picture size.

5 Conclusion

In this project, my goal was to enable website visitors to navigate online news articles using a near real-time web traffic visualization. For each news article, I captured three types of metrics: hits per article, time spent on an article per hit, and the geographic locations of the website visitors. My research on prior work of data visualization helped me incorporate the visual elements in an effective and efficient manner.

In order to test the implemented library, I built a mock news website in Wordpress. I then created a script that constantly emulated random visits to the mock articles. Since I could not reproduce scroll and navigate-away behavior to capture actual read times by using the automated script, I created another script that seeded hits with random times spanning between ten seconds and two minutes. When I hooked my data capture library into the mock website, the web metrics were stored as intended. In the final step, I implemented the interactive web-based visualization using HTML5, CSS3, and JavaScript. Aggregating the data on an hourly basis resulted in a low visual latency, while maintaining temporal accuracy.

I next tested the system with a live Wordpress-based website, **afiadarfur.com** — a website that presents news from Sudan. The capture code successfully captured the hits, read times, and geographic locations into the database.

As for the interactive visualization, the stacked area chart showed around 300 stacked areas for one of the main news categories, which resulted in a considerable latency between changing the date range values in the filter controls box and in seeing the visual update of the chart. This was the result of the news web editors categorizing most of their articles under two main categories, each of which had a high concentration of articles. Thus, for the stacked area chart to redraw with low latency, the editors needed to have spread their articles under several categories such as business, music, world, and entertainment. Adding more filters or narrowing the maximum time window to about one week would have also contributed to lowering this latency.

Clicking on up to two articles resulted in showing the geographic distribution of hits and read times for each article, inside a table. The tabular data was sortable by region name and by number of hits. The number of hits and amount of time spent on an article popped up when the visitor hovered over the corresponding hits bar or attention span circle. The resultant interactive visualization conveyed some interesting observations for **afiadarfur.com**. For example, although the news website's target audience is Sudanese people, most of the traffic was shown to come from Germany and the United States, and none from Sudan.

Up until now, news website visitors were drawn to headlines, titles, and images. With the system that I created, the visitors can go beyond judging the articles by their covers.

6 Future Work

I wish to add more capabilities to my existing visualization. I would like to add a stacked area chart for the parent categories which helps the visitors identify significant peaks in traffic. For example, a peak in the science category could allude to a breakthrough or discovery.

As for articles, I would like to visualize how each article performed relative to other articles, based on the publication date. Another relative measure I would like to incorporate is hits per capita, which results in a more weighted geographic hits distribution. In addition, I would like to increase the ability to simultaneously compare up to four articles.

Lastly, I would like to include social media measures in the existing stacked graph — such as Facebook likes and Twitter tweets. This helps visitors identify patterns in social sharing.

On the web metrics capture end, I would like to implement a cache for the IP addresses of the visitors and the geographic locations they map to. This saves redundant API calls to the third party REST API **telize.com**.

For retrieval of captured data, I would like to implement a cache for the past 31 days of articles' data given a certain category. This minimizes the request load on the database server.

Lastly, I would like to capture each a hit via a token stored in HTML5's **LocalStorage** array. This results in a more accurate hit reading. In this scenario, if two people visiting the same article are behind the same IP address, they would then be counted as two separate hits — versus what is

now counted as one hit. Here, if the visitor decided to disable JavaScript, I could then fall back onto the method I implemented in this thesis — which is identifying each hit as coming from a unique IP address.

7 Glossary

Apache HTTP Server (aka Apache): A web server software program

Choropleth map: A thematic map in which areas are shaded or patterned in proportion to the measurement of the statistical variable being displayed on the map (source: Wikipedia).

Class: In object oriented programming parlance, it refers to the abstraction and representation of a real life object or concept in code.

CSS: A styling language used to define the aesthetics of elements in a web deliverable. This is implemented in all major web browsers. CSS3 is the most recent version of this language.

D3.js: A JavaScript library for manipulating documents based on data using HTML, SVG and CSS (source: d3js.org).

Google Analytics: A service offered by Google that generates detailed statistics about a website's traffic and traffic sources and measures conversions and sales (source: Wikipedia).

HTML 5: The most recent specification for the markup language that constitutes the visual elements of any web deliverable. It also normally includes some JavaScript and CSS 3 code as part of its implementation.

JavaScript: A client-side scripting language that is used by all prominent web browsers.

JSON (JavaScript Object Notation): A lightweight data-interchange format (source: json.org).

LAMP: A web development environment comprised of a Linux server, Apache HTTP Server, MySQL, and PHP.

MySQL: An open source relational database management system.

Object oriented programming: A programming paradigm that represents real-life elements and concepts as "objects". The implementation involves mimicking only the needed real-life characteristics of the object in code, and calling a working copy of these objects an "instance".

PHP: An open source server-side scripting language.

Relational database: A database built on principles of the relational model. Such a database is comprised of tables and fields.

SVG: An XML-based vector image format for two-dimensional graphics that has support for interactivity and animation (source: Wikipedia).

Web visualization: A visual deliverable created using web technologies. More than often, it is dynamic in that it visually changes based on the data being fed to it.

References

[1] Schumann, H. and Muller, W. (2000). *Visualisierung - Grundlagen und allgemeine Methoden*. Springer, Berlin, Germany.

[2] Tufte, E.R. (2001). *The Visual Display of Quantitative Information* (2nd ed.) Cheshire, CT: Graphic Press.

[3] Munzner, T. (2009), *A Nested Model for Visualization Design and Validation*. Retrieved from https://www.cs.ubc.ca/labs/imager/tr/2009/NestedModel/NestedModel.pdf

[4] A. Cockburn, A. Karlson, A.K., & Bederson B.B.(2008). *A review of overview+detail, zooming, and focus+context interfaces.* ACM Computing Surveys (CSUR), vol. 41, no. 1, pp. 1-31. New York, NY: Association for Computing Machinery, Inc.

[5] Playfair,W. and Corry, J. (1801). *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England during the Whole of the Eighteenth Century* (3rd ed.). London, United Kingdom: Wallis.

[6] Frank, A. U. (1998). Different Types of "Times" in GIS. In Egenhofer, M. J., & Golledge, R. G., (Eds.), *Spatial and Temporal Reasoning in Geographic Information Systems*, pp. 40-62. New York, NY: Oxford University Press.

[7] Goralwalla, I. A., Özsu, M. T., & Szafron, D. (1998). An Object-Oriented Framework for Temporal Data Models. In Etzion, O.,

et al., (Eds.), *Temporal Databases: Research and Practice*, pp. 1-35. Berlin,Germany: Springer.

[8] Bettini, C., Jajodia, S., & Wang, X. S. (2000). *Time Granularities in Databases, Data Mining, and Temporal Reasoning* (1st ed.). Secaucus, NJ: Springer.

[9] Radner,W., Diendorfer, G., (2014). English sentence optotypes for measuring reading acuity and speed - the English version of the Radner Reading Charts. *Charts Graefe's Archive for Clinical and Experimental Ophthalmology*, 252 (8), 1297-1303. Berlin; New York: Springer-Verlag.

[10] Wills, G., (2011). *Visualizing time: Designing Graphical Representations for Statistical Data* (1st ed.). New York: Springer.

[11] Cleveland, & W., McGill, R., (1984). Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79 (387), 351-354.

[12] Healey, C.G., (1996). Choosing effective colours for data visualization. *Proceedings of the 7th conference on Visualization* '96, 263-270. Los Alamitos, CA: IEEE Computer Society Press.

[13] Byron, L., & Wattenberg, M., (2008). Stacked Graphs - Geometry & Aesthetics. *EEE Transactions on Visualization and Computer Graphics Archive*, 14 (6), 1245-1252. Piscataway, NJ: EEE Educational Activities Department.

[14] Swires-Hennesy, E., (2014). *Presenting Data: How to Communicate Your Message Effectively*. West Sussex, United Kingdom: John Wiley & Sons Ltd.

[15] Codd, E.F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13 (6), 377-387. New York, NY: Association for Computing Machinery, Inc.

[16] Carpendale, M. S. T. (2003). *Considering visual variables as a basis for information visualisation*. Computer Science TR# 2001-693, 16.

[17] Shneiderman, B. (1996, September). The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on* (pp. 336-343). IEEE.