Visualizing MBTA Data

Mike Barry and Brian Card - June 12th 2014

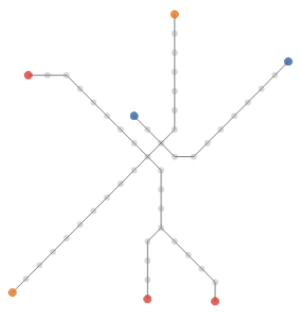
http://mbtaviz.github.io

notes: http://mbtaviz.github.io/handout.pdf

Background

This is an outline of the design and implementation process to create several interactive visualizations using the real time data feed provided by the MBTA and other data sources. The visualizations focus on the performance and behavior of the subway system which differs from traditional train visualizations that focus on the train schedule. We introduce several visualizations characterizing the schedules of the riders on the T, the trains and how the trains interact with each other, and the correlation between congestion and delay on the train system. The main focus is trying to understand the behavior of the system. We tried to answer questions like when and where are the trains crowded or delayed at different points in the system? How do special events affect congestion and delay? How does my commute change from day to day or compare to the commutes of others? This report shares some of the answers to these questions as well as the techniques to come up with those answers.

The Subway System and Data



The MBTA provides data for the red, orange and blue lines, but not green or silver. We used the MBTA's map of the subway system http://www.mbta.com/schedules_and_maps/subway/ as the basis for the minimalist subway map that you see to the right which emphasizes these lines. Data is provided in JSON format from the MBTA and contains the current location of each train as well as the predicted time to the next stop.

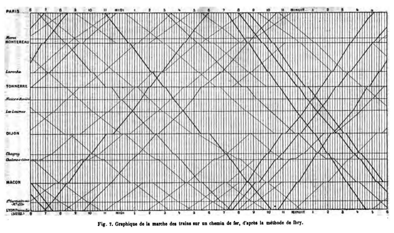
Visualization Tasks

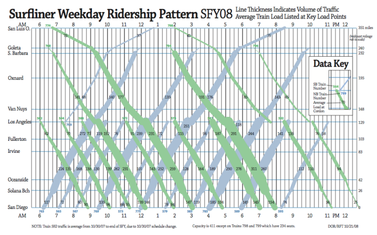
The visualizations focus on answering the following questions:

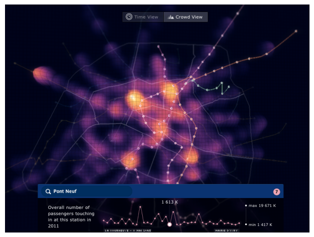
- When and where are the trains crowded or delayed?
- How do snowstorms or special events affect the train system?
- How congested or delayed is my route?

Existing Works

Below is a sampling of existing train visualizations. From left to right, Étienne-Jules Marey's schedule from 1885 [1], Bret Victor's Bart Widget for trip scheduling [2], State of California Department of Transportation report on ridership [3], metropolitan.io a visualziation of the Paris train system [4].







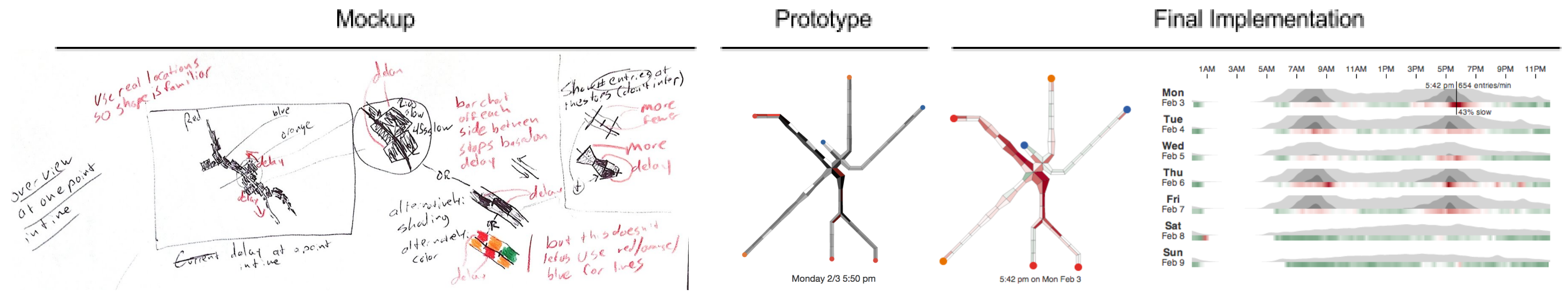
Both Marey and Victor focus on scheduling and not on train system performance. Tufte [5] provides many visualizations of train schedules, all based off of Marey's original design. The State of California also applies Marey's layout and additionally encodes the number of riders on each segment over the year. Metropolitian.io is based of of similar data to what the MBTA provides but uses different visualization techniques.

Key Influences and Design Decisions

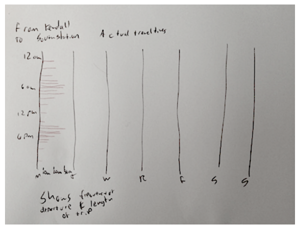
Bret Victor's 2011 essay *Up and Down the Ladder of Abstraction* [6] is a major influence for the design of this project. Victor's insights on how people understand and interact with visualizations, his concepts on understanding systems through levels of abstractions, and even the structure and layout of this essay all served as influences. The visualizations all tie back to maps of the subway which is the viewer's mental model of the system and use tight feedback loops and interactive links inspired by Victor. Tufte was also a major influence, several of his insights such as fully Integrate words, numbers, and images, tips to show mechanism and causality, and suggestions to annotate liberally were taken into consideration in the design process.

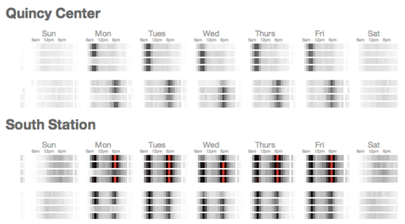
Prototypes

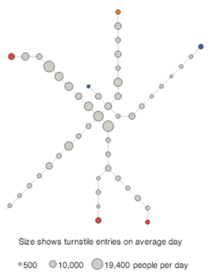
The project started with more than a dozen mock-ups, a sample of which are shown below. Of these 6 to 8 made it into the prototype phase where a sample was made with real data. Prototypes were invaluable in validating ideas and finding trends in the data to explore further. Several ideas such as the congestion and delay visualization only came after significant investment in prototypes.

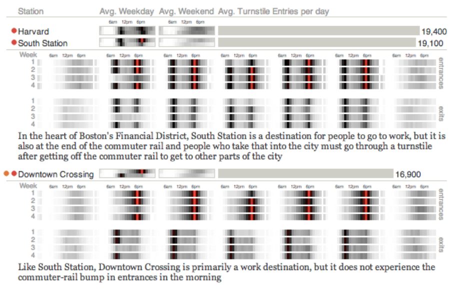


Entrances and Exits mockups, prototypes, and final implementation





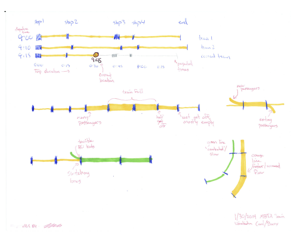


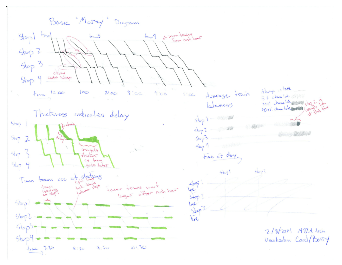


In the heart of Boston's Financial District, South Station is a destination for people to go to work, but it is also at the end of the commuter rail and people who take that into the city must go through a turnstile after getting off the commuter rail to get to other parts of the city

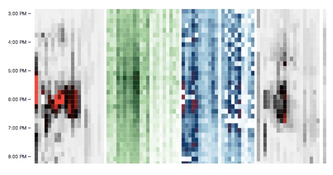
Like South Station, Downtown Crossing is primarily a work destination, but it does not experience the commuter-rail bump in entrances in the morning

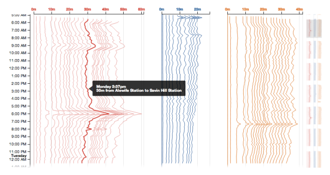
Two of many unused mockups.





Two of several unused prototypes. The prototype on the left show congestion and transit times together which helped us gain an understanding of our data set. The one on the right shows transit times from each stop, a precursor to the Your Commute visualization.



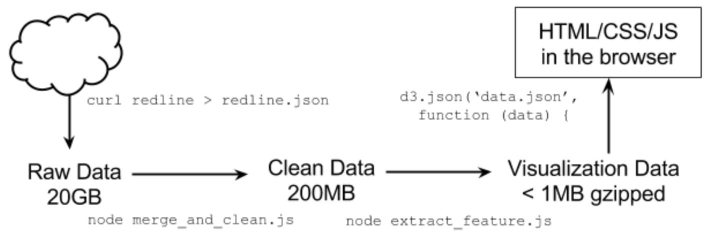


Implementation

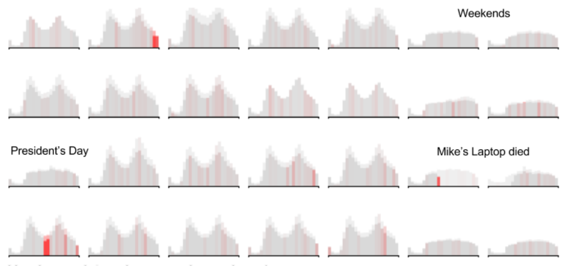
Our visualizations use D3.js to render SVG in the browser. Node.js scripts process the raw data into JSON files which drive the visualizations.

Collaboration Tools

We used bitbucket (https://bitbucket.org/) for private code sharing and task management and GitHub Pages for hosting our compiled static HTML/CSS/JS/JSON files. Google Docs and Google Drive proved invaluable for collaboratively iterating on design/implementation ideas and Twitter was useful for gathering feedback from people we didn't know who share a common interest in data visualization.



Visualization Pipeline



Hourly raw data volume per day and week

Gathering Data

We started trying to gather data using hosted Google App Scripts to pull the data and put it into a Google Spreadsheet every minute. After a few days we found that their service quotas would not allow us to do this for a whole month, so we resorted to using our own hardware. We wrote simple bash scripts and scheduled them using cron jobs on old laptops and ran these in parallel for the month of February to get the realtime data files. We also pulled the MBTA GTFS zip file which contained the scheduling data for all trips and worked with a contact at the MBTA to get per-minute entry and exit counts at each station from their turnstile data.

Merging and Cleaning Data

We experimented with python, scala, and node.js for data pre-processing. Python was an order of magnitude slower than scala or node.js which were comparable. Since only one of us was comfortable with Scala and we both knew JavaScript, we chose node.js for continued work. We found the bulk of the time spent running our scripts was from opening files from disk (12 files per minute of gathered data) so our preprocessing scripts merged the two datasets, removed redundant data, and wrote the data out to hourly gzipped data files with a single json blob per line, for example `redline/yyyy/mm/dd/hh.json.gz`. This gave the optimal performance for downstream feature-extraction scripts, while still allowing us to process date ranges at hour granularity.

Extracting Visualization Data Files

We wrote a node.js script for to generate a minimal JSON data file for each visualization embedded within our larger web page with the goals of minimizing file size and reducing the amount of work done in the browser at runtime. Almost every script ran inside a reusable helper that took a date range, opened the necessary `.json.gz` files and called a function with each line of that file.

Building the Web App

The web application is a single HTML file with descriptions and explanation surrounding placeholder empty `<div>` elements for our visualizations. Our JavaScript files load at the end, read in the data files, render SVG dynamically into the placeholders, and setup listeners to handle interaction. We used a few open source projects to make development easier including D3.js (http://d3js.org) for mapping our data to DOM elements that create the visualization, underscore.js (http://underscorejs.org) for collection manipulation, jQuery (http://jquery.com) for additional DOM manipulation utilities, moment.js (http://momentjs.com) for date and time formatting, and d3-tip (http://labratrevenge.com/d3-tip) for creating simple tooltips. We had had to solve several problems along the way, most of which dealt with scaling common D3 design patterns from a single page rendered from a single data file to a single page with several distinct visualizations rendering data from many data files.

decoupling our visualizations Our primary concern was making sure our visualizations were separate and did not interact in undesired ways when placed on the same page. We wrote each in a separate file and each visualization got its own copy of the data it needed. To ensure our styles were scoped to a single visualization, we used the Less CSS preprocessor (http://lesscss.org), which allowed us to nest selectors and ensure that all rules for a visualization would apply only inside that section of the page. To share any JS and CSS code we explicitly put it into "common" files.

data loading Since we needed to load many data files and render several different visualizations we created a utility that requests several files asynchronously and coordinates progress and completion events, bubbling them up to user-defined listeners

```
VIZ.requiresData([
  'json!data/file1.json', 'json!data/file2.json'
]).progress(function (percent) {
  d3.selectAll(".progress").text('Loading data... ' + percent + '%');
}).onerror(function () {
  d3.selectAll(".progress").text('Failed to load data');
}).done(function (dataFromFile1, dataFromFile2) {
  // use data
});
```

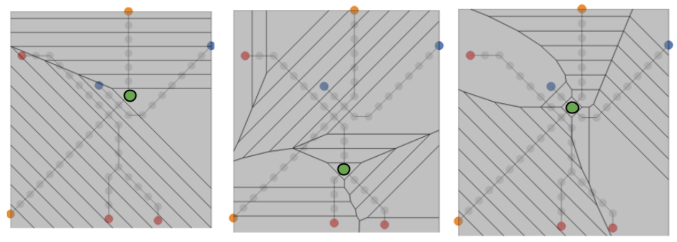
browser compatibility Underscore, D3, and jQuery work across modern browsers and we used an es5/6 shim to let use use newer language features in older browsers. Still, ie8 does not support SVG. Instead of bending over backwards to support them, we used phantomjs (http://phantomjs.org) to render our page to a png file and show that instead.

```
<!--[if lt IE 9]>
  <img src="ie.png">
<![endif]-->
<!--[if gte IE 9]><!-->
  The visualization
<!--<![endif]-->
```

fonts We chose Helvetica for our sans-serif font to match the MBTA's sign style. For our serif font, we wanted to go with Garamond but after some reading learned that Georgia is better since it is optimized for computer screens, whereas Garamond is optimized for print.

linked text Since our visualizations are embedded in a document that describes them, we make heavy use of interactive links in the paragraphs. Most of them highlight the corresponding part of the visualization when you hover, but some of them also change the state of the visualization when you click on them. D3 makes it easy to add event listeners into the text outside the scope of your visualizations for a rich, interactive textual experience.

voronoi picker Our final visualization allows you to click and drag between two stops to see a scatter plot of delay between them. To implement our rules for valid destinations from a source efficiently as you drag, we used D3's voronoi utility which takes a set of points and renders a polygon for each where all points inside that polygon are closest to the point. We render voronoi polygons for the set of valid destinations every time you click and start dragging, and add our mouseover listeners to those polygons as shown below.

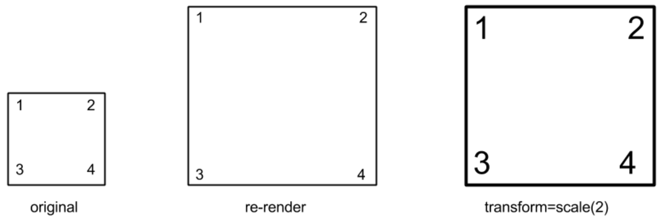


screen size/responsive design We developed on 1200px wide macbooks and optimized for that, but some people we showed this to viewed on iPhones, some viewed on tablets, and Bill Shander viewed it on a large, high-resolution monitor. After this feedback we pulled in Twitter's Bootstrap (http://getbootstrap.com) and customized its responsive design utilities to create 3 responsive width breakpoints:

- <990px which gets a 768px wide page
- 990-1200px which gets a 990px wide page
- >1200px which gets a 1200px wide page

We did not optimize for phones or tablets, but instead set `<meta name="viewport" content="width=768">` in our header to ensure that phones and tablets rendered the page using the minimum fixed width. This allowed us to not worry about rendering our visualizations any narrower than 768px. and also looked acceptable on a range of screen sizes.

responsive SVG using D3? This is tricky, since D3 guides you down the path of using fixed pixel sizes for all of your elements. To make your visualization grow or shrink with the screen size you need to programmatically resize it. A few options are:

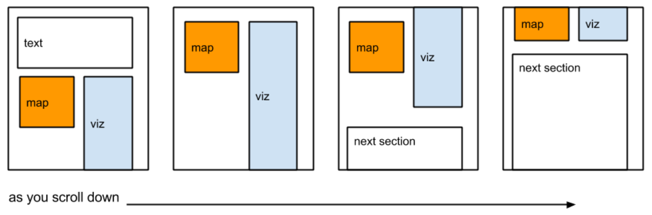


We use the re-render option in most places to preserve text size at different scales, but for the turnstile heatmap which is expensive to render, and the text feels small at the default size, we use the transform=scale(...) option to make our text bigger as well.

scalability & performance As our visualizations grew in size and the size of the DOM increased, we had to optimize a few things to get acceptable performance. Adding mouseover D3 event listeners to thousands of DOM elements took over a second, so we added a utility to setup a single listener to the parent element that handled mouseover events that bubbled up from its children. Calculating new attributes for every datapoint read from a file also got slow, so we pushed those calculations down into our pre-processing layer.

If we wanted to optimize further the first thing to try would be rendering certain parts of our visualization involving many small DOM elements that don't require interaction to canvas instead of SVG. Alternatively, we could just render those section to images and load those instead of rendering each time.

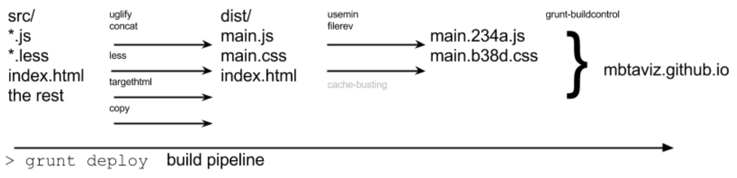
fixing maps in place as you scroll We use scrolling as a means of navigating large visualization, but we wanted to have some fixed context while scrolling through a visualization taller than one page, so we made a utility that listens on scroll events and programatically sets the `top` and `left` css attributes of the map as you scroll to give the effect of the map being part of the document until it reaches the top. Since scroll events fire on iOS devices only when the scroll finishes, we disable this entirely for iPhones and iPads.



Performance was a bit slow on firefox and safari at first, but we added a `transform: translate3d(0,0,0)` CSS style which causes browsers to use hardware acceleration when rendering animations on an element.

Deployment

After developing the application and running it locally, we deploy to GitHub Pages to make it publicly available. This is a free service offered by GitHub where if you push a certain branch your your repository then they will host your static website for free. We use Grunt, a JavaScript build system for deployment as shown to the right. A single `grunt deploy` command from the command line combines and minifies our CSS and JavaScript, rewrites filenames to prevent browsers caching old versions, and deploys the built product to GitHub Pages.



Additionally, Grunt's "watch" plugin provides a development mode where it builds files to a `.tmp/` director and serves them on `localhost` then any time you change a file, it rebuilds those files and forces the browser to reload - this is *very* useful when developing D3 visualizations.

Realtime Version

We experimented with a realtime visualization that polls the MBTA data feed and renders a live visualization of train conditions which is running at http://mbta.meteor.com. Meteor is a self-contained framework that handles live-reloading, minifying, async browser push, and even provides a free deployment service to host your demo on their hardware. This is ideal for a realtime app since our server can poll the MBTA periodically and push updates to the connected clients instead of each client having to poll the MBTA separately.

Successes and Lessons Learned

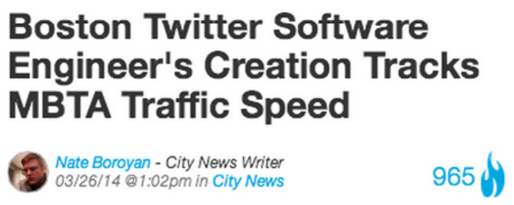
Things we learned about the T:

- mechanical issues account for worst delays
- rush hour causes things to become much more volatile, but on average increased train frequency balances out increased transit times between common stops
- the blue line is rock-solid all the time
- the orange line transit times are rock-solid but their wait times vary drastically
- the red line has the most volatile transit times of all three lines
- trains handle snow storms pretty well

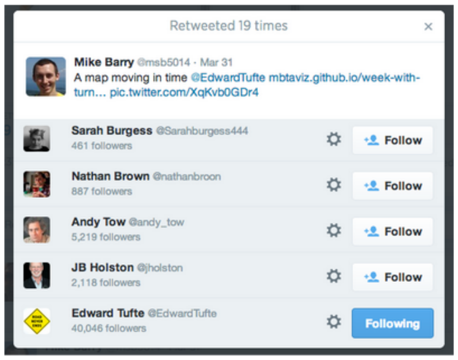
Things we learned about building interactive data visualizations:

- small multiples don't always work on screens as well as on paper since screens have much lower resolution
- knowing D3 is not enough - you also need to know the full breadth of the SVG spec to know what is possible
- mockups and prototypes are good but nothing beats iterating on working code - when you have something concrete to talk to you end up with new ideas you never would have thought of in mockups

Some positive press we received:

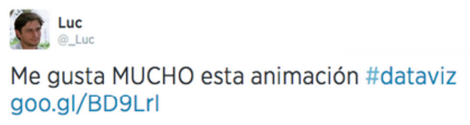

Boston Twitter Software Engineer's Creation Tracks MBTA Traffic Speed

Nate Boroyan - City News Writer
03/26/14 @1:02pm in City News
965

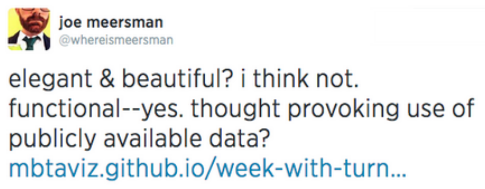

Retweeted 19 times

Mike Barry @msb5014 · Mar 31
A map moving in time @EdwardTufte mbtaviz.github.io/week-with-turn… pic.twitter.com/XqKvb0GDr4

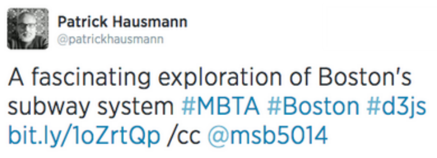
Sarah Burgess @Sarahburgess444 461 followers Follow
Nathan Brown @nathanbroon 887 followers Follow
Andy Tow @andy_tow 5,219 followers Follow
JB Holston @jholston 2,118 followers Follow
Edward Tufte @EdwardTufte 40,046 followers Following


Luc
@_Luc

Me gusta MUCHO esta animación #dataviz goo.gl/BD9Lrl


joe meersman
@whereismeersman

elegant & beautiful? i think not. functional--yes. thought provoking use of publicly available data? mbtaviz.github.io/week-with-turn…


Patrick Hausmann
@patrickhausmann

A fascinating exploration of Boston's subway system #MBTA #Boston #d3js bit.ly/1oZrtQp /cc @msb5014

References

[1] Marey, Étienne Jules. "La Méthode Graphique Dans Les Sciences Expérimentales Et Principalement En Physiologie Et En Médecine. 2. tirage augm. d'un supplément sur le Développement de la méthode graphique par la photographie; avec 383 figures dans le texte." Paris: G. Masson, 1885. https://archive.org/details/lamthodegraphiq00maregoog
[2] Victor, Bret. "Magic Ink: Information Software and the Graphical Interface." Online. 2006. http://worrydream.com/MagicInk/
[3] Nguyen, Lam. "Surfliner Weekday Ridership Pattern." Memorandum to Chairs and Commissioners regarding Follow-up on September and December 2008 Rail Items. State of California Department of Transportation. January 14, 2009.
[4] metropolitian.io. "Metropolitain.io; Paris never sleeps." Online. 2013. http://dataveyes.com/#!/en/case-studies/metropolitain
[5] Tufte, Edward. "Envisioning Information." Graphics Press, Cheshire, CT, USA. 1990.
[6] Victor, Bret. "Up and Down the Ladder of Abstraction: A systematic approach to interactive visualization." Online. October 2011. http://worrydream.com/LadderOfAbstraction/
[7] prefuse.org "Prefuse | Toolkit Structure." Online. August 2007. http://prefuse.org/doc/manual/introduction/structure/