CS462 Final Project Report
Aaron Monson, Curt Merrell
BYU Parking Gossiper

Problem:

Students spend a lot of time circling parking lots trying to find parking spots during high traffic times. This often causes them to be late to class, as they travel to distant parking lots when spaces may be available closer to their classes.

APIs (component/external and internal):

- **External APIs**

  Twitter - /statuses/update
  - Our system runs nodes on a server that receive “Car Entered” and “Car Exited” push events. The easiest way to provide live updates for users without having them download a new app is to push availability statuses straight to twitter.
  - Most students already have twitter, or can download the mobile app for it easily. This made it so we did not have to write a UI for general users.

  Twitter - OAuth
  - In order to post statuses to twitter, a user has to be authenticated. This means we had to research Twitter’s oauth API and figure out how it works. This was simply necessary, and consumed a large amount of our time on this project.
  - Each node runs on some port of the server that is TLS encrypted using openssl. At startup, the admin tells the server the location of a config json that contains any necessary keys for the twitter account attached to that node.

- **Internal APIs**

  /carEntered
  - Endpoint signaling a car entering the parking lot.
  - Currently we fire this event from an admin simulation UI because the hardware to signal real changes was not in the scope of our project.
  - This endpoint would be called by the tripwire once the tech is installed and would work seamlessly

  /carExited
  - Same basic function as the carEntered endpoint, except it signals that a car has left the parking lot.

  /:nodeUrl
  - Connects the current node (calling this endpoint) to a new node for gossiping as specified by the node url.

Actors:

- **Tripwire**

  Pushes “carEntered” and “carExited” commands to the parking lot they are connected to.

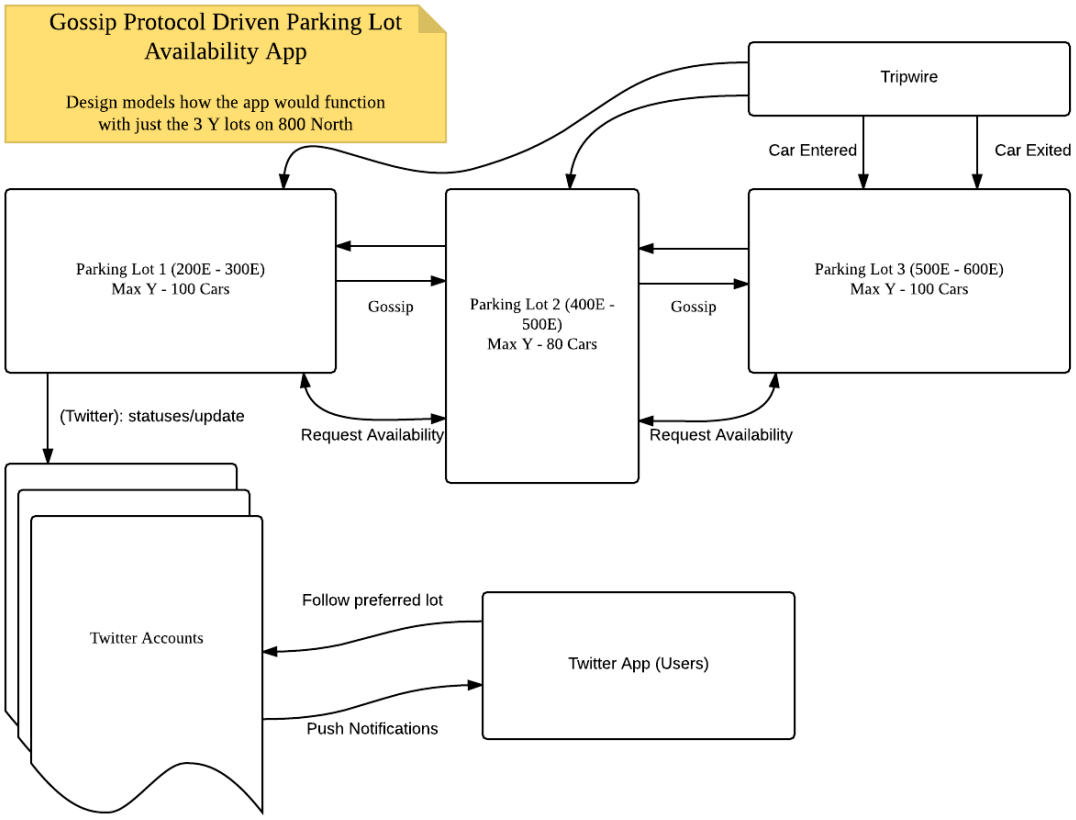
These tripwires don't actually exist, but we simulated them in such a way that they can be added seamlessly.

- **Parking Lot**

  The most essential actor in the model. Each parking lot is connected to an account on twitter and posts availability updates when the availability in a parking lot has changed significantly.

  - Lots also gossip with their two nearest neighbors.
    - When a lot is full, it will link followers to it's nearest neighbor that has spots available. (see https://twitter.com/800n500eProvo)
  - Lots will currently never add more than 2 neighbors to their list of gossip recipients, so the model was testable with only 2 nodes.
  - Lots never send want requests. Because only the most recent rumor matters, we did not need to worry about rumor history.
    - If a lot node goes down, it will just re-broadcast its availability when it comes back up.

The below diagram illustrates how these actors interact with each other.



- **Twitter**

  Twitter receives pushed “statuses/update” commands that send a body of text to display updates from the app’s parking lots.
    - All user interface happens through the Twitter UI. We did not need a custom UI for the purpose of our app.

- **Users**

  Users follow one or more parking lots’ twitter accounts to receive status updates when relevant availability changes occur

Diagram and Architectural Explanation:

Our previous bullet points describe the bulk of the architecture and include the primary diagram of events and actors as well. As a summary:

- All parking lots exist solely as a node on a server.
- Tripwires push “carEntered” and “carExited” commands to a parking lot they are attached to.
- Parking lots post status updates to twitter if significant changes have occurred since their last tweet.
    - For us, this was a change of 5 or more spot availability, or any change from full to not full or vice versa.
- Users can follow one or more twitter parking lot accounts as desired to be notified of updates in real time.

Methods Used:

Our server runs in nodejs with a small front-end developed in angular. The front-end is used just to connect parking lots to each other and manually fire carEntered and carExited commands, so it is lightweight and intended only for admin use.

Twitter apis were requested using a node package called “Request”. All requests try to follow basic REST architecture, but sometimes deviate to accommodate a more event-driven architecure.

We implemented another node package called node-twitterbot. This package stores relevant Twitter REST api information for compact execution whenever needed.

Analysis:

An event-driven model was perfect for the application we wrote. Cars enter parking lots, and cars leave parking lots. Each of these is a pretty basic event, so all of the functionality that stems from these events was easily translatable into extended events.

Pros:

- A natural fit for the problem being solved
- Makes an event-driven “push, not pull” architecture very easy to implement
- Code is simple to follow for a small project

Cons:

- Sometimes the event-driven nature of the app made our implementation tricky, since the bulk of our endpoints followed REST architecture
    - Tripwires fire synchronous events, but the subsequent twitter calls used asynchronous REST architecture. There were a few complications here, but not many.

Overall, our project fit almost perfectly to an event driven architecture.