Aperator: Making Tweets Enable Actionable Commands on Third Party Web Applications

Peter Zakin
Princeton University
pzakin@princeton.edu

Soumya Sen
Princeton University
soumyas@princeton.edu

Mung Chiang
Princeton University
chiangm@princeton.edu

Abstract

Twitter has become a persistent part of our digital lives, connecting us not only to our individual audiences but also to an entire landscape of applications built for the web. While much has been done to support the Twitter ecosystem outside of Twitter, little has been done within Twitter to power those same applications. This work introduces a service called *Aperator*, which supports application-specific actionable commands through tweets. This ability creates several interesting opportunities for both end-users and application developers building on the Twitter platform. For example, the actionable command capability allows a link that a Twitter user shares with his followers to be directly added to any of the user's connected link sharing networks, such as Delicious or Read it Later. The client side of this system has a console for end-users to sign up and provide their login credentials for various web services that our system supports: Delicious, Foursquare, Read it Later, Foursquare etc. The system's backend has two cron jobs that run every minute to: (a) retrieve and parse tweets from a specific twitter account and store them in a command form in a MySQL database, and (b) execute the unexecuted commands found in the users tweets. This paper describes the concept, implementation, and results from an experimental study of this new application.

1 Introduction

Since its launch in 2006, Twitter has become one of the most important social properties on the web, trailing only Facebook and YouTube in terms of traffic [1]. Beyond the popularity of its own service, however, Twitter has also promoted the growth and engagement of third party websites through its API. As of May 2011, there are over 660k developers building applications on the Twitter API and over 900k operating applications [5]. The Twitter API has four major offerings: Twitter for Websites enables visitors at third party sites to make use of basic Twitter functionalities like following and tweeting on third party sites; the Search API provides query-access to recent Tweets; the REST API provides a way for developers to access user data and execute most of the main functionality of the Twitter service; lastly, the Streaming API permits an uninterrupted connection to the Twitter Firehose for developers to make use of data-sets.

Generally speaking, consumer web applications predominantly use the REST API and do so in three primary ways: to publicize user activity on the Twitter network (e.g. Foursquare, Quora), to stream user activity (e.g. Summify), or to operate their own Twitter client (e.g. TweetDeck). But the service that this paper describes represents a new mode for relating consumer web applications to Twitter.

The motivation for this approach is that although today's social web is great for sharing [8, 9], with so many apps it can be hard to connect what we share in one network to our presence on others. Moreover, for those who spend most of their time on only the major networks, it can be difficult to keep up with their audience on others [7]. Some networks today do provide limited interactivity through automatic forwarding of all user updates to a few other networks, as shown in Figure 1.1. But instead of either forwarding all updates from one network to the others or needing to log in and post on multiple networks, what if a user had the ability to selectively post from one network to all the other networks that the user cares about? Aperator [2, 3] realizes this vision by enabling users to operate real commands on a set of different web apps just by tweeting.

1.1 Contributions

Aperator demonstrates a new means of posting, which creates numerous benefits including:

- A method for *granular* cross-network posting: Cross-network posting is currently possible from

Twitter to Facebook and from Foursquare to Twitter, among others. But this form of cross-network posting is automatic and applies to all posts or none at all. The selective nature of aperator commands makes cross-network posting granular.

- A solution to make *multi-network online presences* more convenient for users: Although users may maintain several presences across a variety of web properties, it can be difficult to actively contribute on all of them. Generally and not at all surprisingly, users spend most of their time on larger networks like Facebook and Twitter as opposed to smaller networks like Delicious or Read it Later. Since aperator enables users to post content on some of those smaller networks from Twitter, it makes the multi-network presence more plausible.

- A way to increase *engagement* for third party applications: By making posting on third party applications as simple as tweeting, users can broadcast to multiple networks through one single interface. Thus, aperator can boost engagement on third party applications as another avenue for posting.

- A new *platform* for application development: Since users can interact with third party applications through the Twitter interface, aperator demonstrates the possibility of purely back-end applications. As an example, we created a prototype for such an application built on top of the aperator platform. The application, called "sms", provided the basic functionality of a group text messaging application from Twitter by using Twilio's service. Users could sign up by logging into aperator and editing their sms settings, which entailed adding or editing groups of numbers. If users wanted to send a text message to the members of their sms-groups, they would tweet: "@aperator sms #GROUPNAME Message".

Although increasing connectivity among different web applications in itself is not a new idea, Aperator's key contribution is in demonstrating what this system can enable. Aperator can be seen as a first step towards a TwitterOS, and using Twitter to provide some interesting features and services as discussed below.

First, there is a familiar and easy to use interface and you can access it from a wide range of clients. As a result, we did not develop a separate stand-alone shell-client on *aperator.com* and instead let users interface directly through Twitter.

Second, when users start using aperator services extensively and more features are developed by third-party developers using aperator as a platform, then user's command history will be publicly available for innovating a

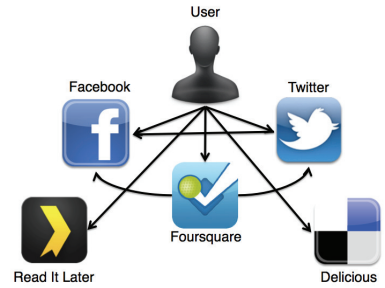

Figure 1: Partial interactivity among different social network applications

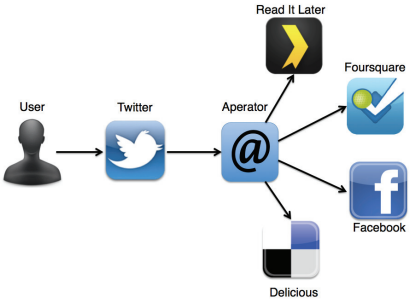

Figure 2: Aperator allows users to tweet selectively to audiences on different networks

variety of new services. For example, users can benefit from searching and finding out how their fellow users are using various sequence of commands to accomplish some particular task. While there are privacy concerns to be addressed, the presence of such a platform can create new possibilities for technical innovations.

Third, while Twitter already permits in-tweet commands for cross-posting to LinkedIn and Facebook, this capability stems from coordinated relationships between Twitter and other applications. Aperator democratizes this capability, making cross-posting simple even for organizations that are smaller than Facebook or LinkedIn. Additionally, Aperator commands have the potential to execute more sophisticated operations than posting, while Twitter's current model for cross-posting seems ill-poised to do so.

It should be noted that Aperator is not a 'tweeting from the command line' application like Twidge, Twitter CLI etc; it is effectively accessible from any Twitter client. Also, since Aperator is not a client interface, the recent moves by Twitter to discourage third party Twitter client developments in favor of consistent user experience [6] is not at odds with our approach.

1.2 Approach

Aperator [2] is a platform for making tweets actionable on third party web applications. By tweeting to the @aperator account, users are able to post content on web applications like Facebook, Foursquare, Delicious and

Read it Later, as illustrated in Figure 2. Just as a command line connects its users to applications on their operating system, Aperator connects its users (through Twitter) to applications on the social web. And like the command line, Aperator operates with a strict syntax, even as it enables powerful capabilities through a simple interface. The syntax currently supports four commands:

1. Link submission to Delicious: "@aperator delicious www.example.com [optional text]".
2. Link submission to Read it Later: "@aperator ril www.example.com [optional text]".
3. Post a status update on Facebook: "@aperator fb This is a status update".
4. Check-in on Foursquare: "@aperator 4sq Example Location".

In the absence of a revenue source to pay for users' texting, the sms app has not been featured in the public release of Aperator, but it does demonstrate a potentially powerful model. Building the sms application was greatly simplified since the user interface was almost entirely located on Twitter – the only exception was the group set-up page, on which users created groups and specified the names and numbers of its members. This shows that application developers can build mostly back-end applications that require little to no front-end interface since the end-users tweets have been shown to be sufficient as a means for user input.

This paper is organized as follows: Section 2 describes the client-side and back-end architecture of Aperator. The implementation details, challenges, and limitations are elaborated in Section 3. Initial performance results from a small-scale functionality test of the system is reported in Section 4. Section 5 discusses the potential of app-specific command execution capability and future extensions, followed by conclusions drawn in Section 6.

2 Architecture

2.1 Client-side design

To get started on the service, users are required to sign up with their Twitter credentials. After Twitter authentication, users are redirected to aperator's signup page, where they also create their own Aperator login credentials. When users are logged in, they are presented with a visual display of aperators "Lexicon", which lists the available applications that can be connected alongside their associated command-syntax, like in Figure 3.

In order to start using any of the four apps that are currently supported by aperator – Foursquare, Delicious, Facebook and Read it Later – users first connect them to

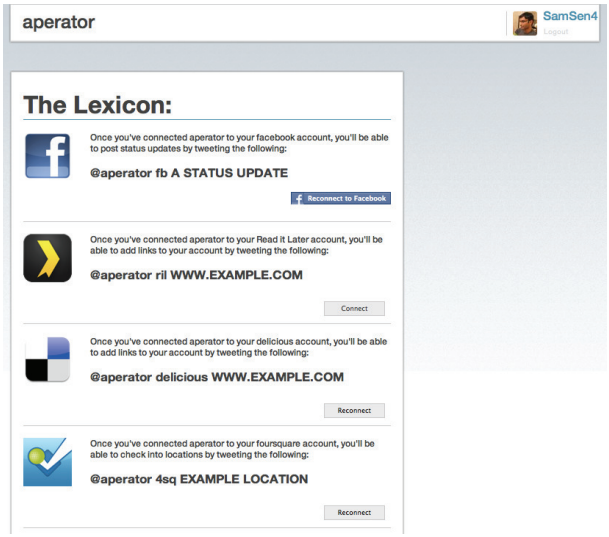

Figure 3: Signup screen with a list of supported web apps

Aperator. For users wishing to connect aperator to their Foursquare or Facebook accounts, users follow a connect button to either apps' oAuth 2.0 authentication process, which upon verification, will redirect the user back to Aperator. But because Delicious and Read it Later utilize HTTP-Auth, Aperator stores users login credentials for these services. Therefore, when users choose to connect either of these two services to aperator, they are taken to a connect page. Upon successful connection, users are redirected back to the aperator home screen.

After users have connected aperator to any of the available apps, they can reconnect to an app if they changed their login credentials or if they happened to inadvertently revoke access to aperator. To reflect the ability to reconnect an app with different login credentials, what was previously labeled a "connect" button is now labeled for connected users as "reconnect."

2.2 Back-end design

While the end-user interface occurs almost entirely on Twitter's cross-platform properties after signup, the implementation and core of the system runs on the services server, which runs as an Amazon Web Services instance. The back-end processes involve two main parts: storing tweet-commands and executing commands that have not been executed.

Tweets are stored through two cron processes that run every minute, called stream1.php and stream2.php. Both processes are identical but for a sleep cycle in stream2.php which lasts 30 seconds. This way, tweets are captured from Twitter twice every minute – nearly every 30 seconds. The frequency of tweet-capturing can be increased as the service's adoption grows. Although Twitter does not release a formal rate-limit for the Search

API, the risks of being throttled seem to advise a conservative approach to tweet queries.

It should be noted that using the Streaming API is more conducive to the demands of this application since it allows developers to maintain long-lasting connections to the Twitter Firehose. In future releases, the Streaming API should be utilized since it would relinquish the need to run multiple cron processes and concerns about rate-limiting. However, given the prototype nature of the current version, we presently capture tweet-commands using the Search API.

The following snippet demonstrates the cURL response used to query the Search API:

```
curl http://search.twitter.com/search.json?q=\%
40aperator&include_entities=true
```

The search is specifically for all @mentions of @aperator, specifying tweet-entities as the return type. Specifying "include_entitites=true" in the request asks Twitter to include the expanded URL of links that Twitter has converted to the t.co link-shortened format. This request to Twitter returns a JSON response:

```
{"result_type":"recent"},
"profile_image_url":"...., ....,
"text":" @aperator ril http://t.co/gzPGiehI",
"to_user":"aperator",
"to_user_id":427607438,
"to_user_id_str":"427607438",
"to_user_name":"Aperator",
"created_at":"Wed, 11 Jan 2012 19:27:43 +0000",
"from_user":"pzakin", ....
```

The JSON specifies the constituent elements of a tweet object, e.g. "to_user_name" and "text", and is easily parsed. From here, the "from_user" property of the tweet is checked against a MySQL table of Aperator users in order to make sure that tweets from non-users will not be processed. Assuming that the tweet comes from a registered user and that the app specified is valid, the command is stored in a MySQL table. The second half of the implementation consists of a process that executes commands that are yet to be executed. This process is managed by a cron job that runs each minute. To maximize for speed of execution–i.e., limiting the lag between catching the tweet from the Twitter Search response and its execution–the execution process operates inside a while loop, which iterates not less frequently than every five seconds. Upon execution, the command is designated as executed and ignored by the executing process in the future. Figure 4 shows the overall architecture and the processes that constitute the aperator system.

3 Implementation

The client console was developed using basic HTML, CSS and Javascript–the latter of which was mostly implemented using the javascript framework, jQuery. The back-end was built on a LAMP stack that was hosted on an EC2 instance from Amazon Web Services. As has already been described, the system itself was powered primarily by the Twitter API, which connected the aperator application to users' tweets.

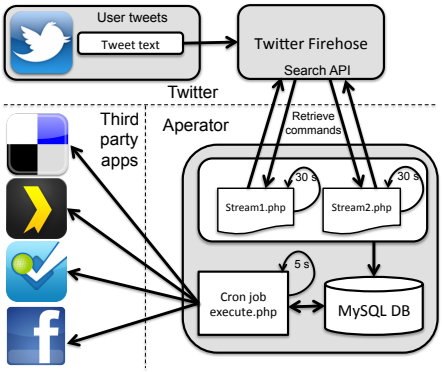

Figure 4: Aperator architecture

Although our current initial prototype uses the Search API, the use of Streaming API and even the REST API are also feasible. The Streaming API represents the best method for accessing @mentions of @aperator because it streams tweets almost instantaneously. We chose to use the Search API in the current prototype merely to avoid potential complications with maintaining a long-standing HTTP connection. Additionally, we opted against the REST API because of its requirement for authenticated use of the API.

3.1 Limitations

Although the Search and Streaming APIs can rapidly stream @mentions of @aperator, both cannot access @mentions coming from private users. Therefore, Twitter users who maintain a private account viewable only to their followers are presently unable to use the service.

Going forward, we have considered improving our current implementation by utilizing the Streaming API to catch @mentions of @aperator instantaneously. Additionally, to provide service access to Twitter users with private accounts, we will add a process utilizing the REST API that should run in parallel to our Streaming API connection. Since the REST API requires authentication, private @mentions will be retrievable as long as we require users to follow @aperator when they sign up, which can be easily added to our signup process.

Additionally, the current implementation for checking in on Foursquare merely "shouts" a venue name. We do not specify a 'venue id' that would enable a traditional Foursquare check-in. In order to retrieve a 'venue id', we would need to specify latitude and longitude coordinates. In fact, this should be possible since users can add their

location to their tweets. Unfortunately, at this time, the Tweet entities returned by the Search API have returned "NULL" values for the "geo" field that otherwise should specify the needed coordinates. Until this issue has been resolved by Twitter, we are unable to check-in users to registered Foursquare locations.

3.2 Challenges

Besides the issues discussed regarding the Foursquare client, it is worth mentioning some other challenges and the workarounds we undertook to help coordinate access to Delicious and Read it Later in a user-friendly way.

First, Twitter shortens all links to the t.co wrapper format. When links are added to Read it Later or Delicious, it is important that they are translated back to an expanded URL. Otherwise, users viewing their saved links would fail to recognize the destinations of their links. Since Twitter provides an "expanded_url" field in their JSON response to search queries, we use it so that when the expanded_url is valid, the delicious request adds the expanded URL rather than the t.co shortened link.

Lastly, the Delicious API requires a "description" parameter for link-submission. In order to provide a meaningful description, we decided to scrape its "<title>" tag from its DOM structure. The code we used for that purpose is given below:

```
function scrapeTitle($url)
{   $file = file_get_contents($url);
    preg_match('/<title>(.*)<\/title>/i',
    $file, $title);
    $description = trim($title[1]);
    return urlencode($description);}
```

4 Evaluation

4.1 Deployment

The service website [2] was formally opened up to users on January 2, 2012 for experimentation and testing. Since then, 47 users have signed up for the service and 82 commands have been issued from Twitter, as shown in Figure 5. Although the adoption has been slow in the absence of promotional efforts on our part for this non-commercial service, our focus has been on testing and improving the system functionality as opposed to enlarging user base. Going forward, it will be important to reach out to Twitter users and adapt the service based on user feedback. Increasing the number of applications and services that users can connect to from Aperator will also be explored in the future.

4.2 Performance

We evaluate the performance of the system by measuring the execution time of commands issued to Aperator from

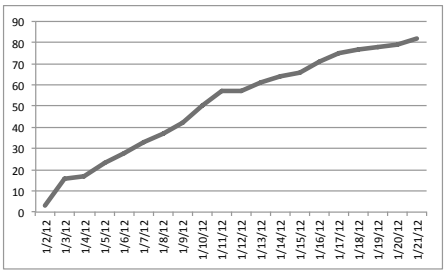

Figure 5: Cumulative growth in tweets sent to Aperator

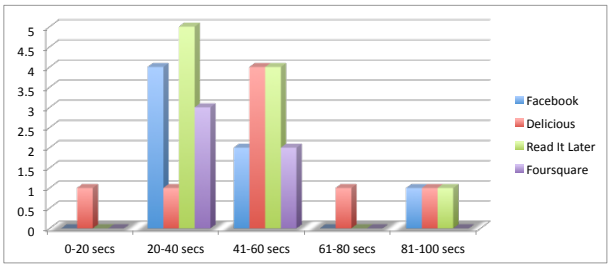

Figure 6: Histogram of the execution time of 30 commands for different web apps

user tweets. Each time a user submits a tweet, we record the time at which the tweet was created as well as the time at which the command was executed. Thus, execution time measures the time taken from a tweet post on Twitter to its execution for posting on another network. Based on a sample of 30 tweets, with execution times ranging from 20 seconds to 94 seconds, we found the mean execution time to be 44.23 seconds. The estimation of execution times for tweets of different third-party services is shown in Table 1.

Figure 6 shows a more granular picture of the distribution of execution times for tweet commands operated on by Aperator for different applications. The performance figures show that Delicious and Read it Later commands have relatively slower execution because both implementations entail an additional process of scraping titles for submitted links, which we've detailed before.

5 Discussions

Despite the presence of APIs across the vast majority of popular web services, the flow of data generally moves towards the larger networks as smaller networks publicize activity to a greater audience. In a sense, Aperator is a serious step towards moving more data from the larger networks to the smaller ones–which in effect, amounts merely to the ability to post content from a larger network to a smaller one. Because users spend more time on larger networks, Aperator effectively lowers the barriers to engagement that plague smaller networks.

While we currently support commands only coming from Twitter, it is possible and perhaps advantageous to replicate the Aperator service for Facebook. Sending

Table 1: Command Execution Time Statistics

Application	Mean (secs)	Std. Dev. (secs)
Delicious	50.25	20.58
Read It Later	43.4	21.72
Facebook	44.29	24.02
Foursquare	36.2	10.98

commands to third party web applications from Facebook would work quite similarly to the Twitter implementation. Users would merely update their status with specific commands for third party web applications. Admittedly, this would require a syntax adjustment because Facebook does not support user-to-application communication. Thus, to specify an Aperator command in Facebook, we plan on introducing a syntax such as: "## delicious www.example.com".

5.1 Related Developments

Increasing connectivity among different web applications is not a unique idea, even though Aperator is in many ways a unique implementation. IFTTT [4], for example, works with a host of different applications to automate actions between different services. For instance, if one uploads a file to her dropbox folder, IFTTT might send a tweet or a text message or post a status update on any number of services etc. The range of "recipes" permitted by IFTTT is extremely compelling when considering the potential benefits of inter-app synergy.

IFTTT has become a popular service in a new category of applications we might well call "API plumbing," but it certainly need not be the last. Aperator differs from IFTTT in the granularity of control it offers and the dynamism with which users can specify executable commands directly as tweets. There is tremendous potential to improve and rethink the relationships between different apps: synergy paves the way for new, unexplored experiences in the consumer web. Some of those experiences that deserve further exploration are presented next.

5.2 Future Use-cases

Facebook and Twitter have popularized the idea of signing up for web services using their respective login credentials. Supporting Facebook and Twitter authentication on third party applications expands the reach of their networks and for the benefit of third party properties, simplifies the authentication process and increases engagement. One of the ways in which Aperator increases the accessibility of third party web applications might very well be in authentication. Instead of signing up with Facebook or Twitter, users could simply tweet: @aperator install APPNAME. Once again, the analogy of the command line serves inspiration well. For just as "yum install emacs" provides a simple model for package installation on the command line, "@aperator install APPNAME" could provide an easier way for users to register on third party web applications as Aperator funnels authentication tokens in a pipeline from Facebook or Twitter to another application.

Right now, the Twitter presence of web applications amounts to little more than a stream of updates and announcements. But through Aperator, applications can begin to have actionable presences inside of a network. The suggestion, begged by the use-case, may be that Twitter could function in parallel to HTTP as an application medium. There is some truth, then, to the words of Paul Graham, the founder of Y Combinator, who wrote: *"Successful new protocols are rare... So any new protocol is a big deal. Each one of those protocols has spawned many successful companies. Twitter will too."*

Following such thoughts, a package installer for web applications is just the tip of the iceberg when it comes to new apps that can be developed on the Aperator platform.

6 Conclusions

This work introduces a new idea and an initial system prototype for a service, called Aperator, which supports application-specific actionable commands through tweets. Aperator facilitates granular cross-network posting and increases user convenience, thus opening up avenues for greater social utility and interactivity across web services. We review several benefits of this approach for both end-users and third-party applications, provide an architecture for enabling such services, report on initial performance results, and outline several potentially interesting extensions of this system.

References

[1] Alexa. http://www.alexa.com.

[2] Aperator. http://www.aperator.com.

[3] Aperator Demo. http://www.youtube.com/watch?v=tBzqShO29Xw.

[4] IFTTT. http://ifttt.com/.

[5] Twitter. http://www.twitter.com.

[6] BROWN, M. Twitter Clamps Down On Third Party Clients. Wired [online], 14 March 2011. http://www.wired.com/epicenter/2011/03/twitter-third-party-clients/all/1.

[7] BRUNO GONCALVES, NICOLA PERRA, A. V. Validation of Dunbar's number in Twitter conversations, May 2011. arXiv:1105.5170v2.

[8] KRISHNAMURTHY, B., GILL, P., AND ARLITT, M. A few chirps about twitter. In *Proceedings of the first workshop on Online social networks* (2008), WOSN '08, pp. 19–24.

[9] KWAK, H., LEE, C., PARK, H., AND MOON, S. What is Twitter, a social network or a news media? In *Proc. of the 19th int'l conference on World wide web* (2010), WWW, pp. 591–600.