Great East Japan Earthquake Viewed from a URL Shortener

Takeru Inoue
JST ERATO Minato Discrete Structure Manipulation System Project
North 14 West 9, Sapporo 060–0814 Japan
takeru.inoue@ieee.org

Fujio Toriumi
Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464–8601, Japan
tori@is.nagoya-u.ac.jp

Yasuyuki Shirai
JST ERATO Minato Discrete Structure Manipulation System Project
North 14 West 9, Sapporo 060–0814 Japan
shirai@erato.ist.hokudai.ac.jp

Shin-ichi Minato
Hokkaido University
North 14 West 9, Sapporo 060–0814 Japan
minato@ist.hokudai.ac.jp

ABSTRACT

On March 11th 2011, a great earthquake and tsunami hit eastern Japan. After that, several web sites, especially those providing helpful disaster-related information, were overloaded due to flash crowds caused by Twitter users. In order to mitigate the flash crowds, we develop a new URL shortener that redirects Twitter users to a CDN instead of original sites, since Twitter users rely on URL shorteners like bit.ly to shorten long URLs. In this paper, we describe our experience of developing and operating the URL shortener in the aftermath of the giant earthquake. Since the flash crowds were a serious problem in an emergency, we had to develop it as quickly as possible with a spirit of so-called agile software development. We then explain our HTTP request log collected at the URL shortener (it is now available online). To investigate the cause of flash crowds, the log is examined with tweets (Twitter messages) provided by another research project; this collaboration is realized by the encouragement of the workshop committee. We hope our experience will be helpful in tackling future disasters.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General; H.3.5 [**Information Storage and Retrieval**]: Online Information Services, Web Based Services

ACM SWID 2011, December 6, 2011, Tokyo, Japan.

General Terms

Experimentation, Measurement

Keywords

flash crowd, URL shortener, Twitter, CDN, disaster, Great East Japan Earthquake

1. INTRODUCTION

After the Great East Japan Earthquake, several web sites providing disaster-related information failed repeatedly, due to *flash crowds* caused by Twitter users. Though some sites increased their capacity with cloud technologies or distributed their loads among CDNs, it is difficult for most sites to react quickly in the aftermath of an earthquake (some site administrators might be stuck in shelters themselves). For flash crowd mitigation, we drop a tweet that suggested to replace URLs in a tweet with those of CoralCDN, a free content distribution network [16]. In the chaos following a disaster, however, it could not be expected that people would stay cool enough to replace URLs before posting messages.

In order to mitigate the Twitter flash crowds, we gave attention to *URL shorteners*, which are often used among Twitter users to offset the strict limit on message length. Our basic idea is that URL shorteners have a potential to divert flash crowds to CoralCDN from the original site. Moreover, the use of URL shorteners is transparent to the Twitter user, and so manual replacement of URLs is not needed.

We began to develop a new URL shortener on March 13th, just two days after the earthquake, with a couple of energetic collaborators (shown in the acknowledge-

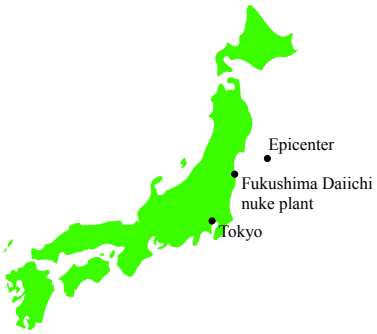

Figure 1: Map of Japan showing the epicenter of the earthquake and the Fukushima Daiichi nuclear power plant.

ment). Since the URL shortener should be launched as early as possible, we implemented its features in an incremental manner. Our incremental approach in the development worked very well, and we could deploy a prototype in the next day. Our URL shortener was named *d.x0.to* in the prototype, and was renamed *rcdn.info* in the stable version. In this paper, we describe our development history in detail.

This rapid development also allowed us to collect the HTTP request log in the emergency at the URL shortener; it is now publicly available online [7]. The dataset is not that large, but we believe that its information is extremely valuable in capturing user behavior in an emergency. In this paper, we see the dataset from some aspects, and examine it with tweets in terms of flash crowds[1].

The rest of this paper is organized as follows. Section 2 reviews the Great East Japan Earthquake, URL shorteners, and CoralCDN. Section 3 overviews our URL shortener. Section 4 describes our development history, and Section 5 explains the public dataset. In Section 6, we examine the dataset with tweets and discuss flash crowds. Finally, Section 7 concludes this paper.

Along the spirit of the workshop, we would like to focus on sharing our experience including development history as well as interesting dataset, and details of the system design and log analysis will be discussed at some other place.

2. BACKGROUND

2.1 Great East Japan Earthquake

A great earthquake of magnitude 9.0 occurred off the eastern coast of Japan (Fig. 1), at 14:46 JST on March 11th 2011. It was the most powerful known earthquake to have hit Japan, and one of the five most powerful earthquakes in the world since modern record-keeping began in 1900. The earthquake triggered extremely destructive tsunami waves along the eastern coast of Japan. In addition to the significant loss of life and destruction of the infrastructure, the tsunami caused a number of nuclear accidents at the Fukushima Daiichi nuclear power plant; several hydrogen explosions at the reactors scattered low level radioactive materials around the local area. The nuclear incidents greatly disturbed people living in a wide area that included Tokyo. They rushed onto the web searching for helpful information and this public action formed flash crowds.

2.2 URL Shorteners

The first URL shortener, "Make A Shorter Link", was launched in 2001 to eliminate the frustration of copying very long URLs [10]. Currently, URL shorteners are mainly used by short messaging services like Twitter, which severely limit the number of characters in a message. A URL shortener, which usually has a short domain name, generates a unique alphanumeric key for each URL. The domain name and unique key form a short URL like `http://bit.ly/v1Anp`, which redirects visitors to the original URL. A substantial amount of web traffic currently goes through URL shorteners. One report indicated that short URLs on bit.ly, the largest URL shortener, were accessed 2.1 billion times in November 2009 [3].

Figure 2 shows an example of URL shorteners, which includes *shortening* a long (original) URL and *expanding* a short URL.

1. **Shortening original URL.** A client (a smart phone in the left-hand side in the figure) makes an HTTP request for shortening a long URL (e.g., `http://example.com/path`). Upon receiving the request, the URL shortener generates a unique key (e.g., `v1Anp`) for the URL, and stores the key with the original URL in its database. The URL shortener, then, responds with the short URL (e.g., `http://bit.ly/v1Anp`).

2. **Spreading URL.** The short URL is spread among Twitter users by *retweeting*.

3. **Expanding short URL.** Clicking this short URL, a client makes an HTTP request for the short URL. After receiving the request, the URL shortener retrieves the associated URL from its database. The shortener, then, returns a response with the original URL.

4. **Causing a flash crowd.** Finally, the client is redirected to the original URL. Since retweeting allows users to quickly spread their favorite tweets,

[1]The tweets were collected by another research project [18], but we have a chance of collaboration with the project thanks to the programming committee of this workshop.

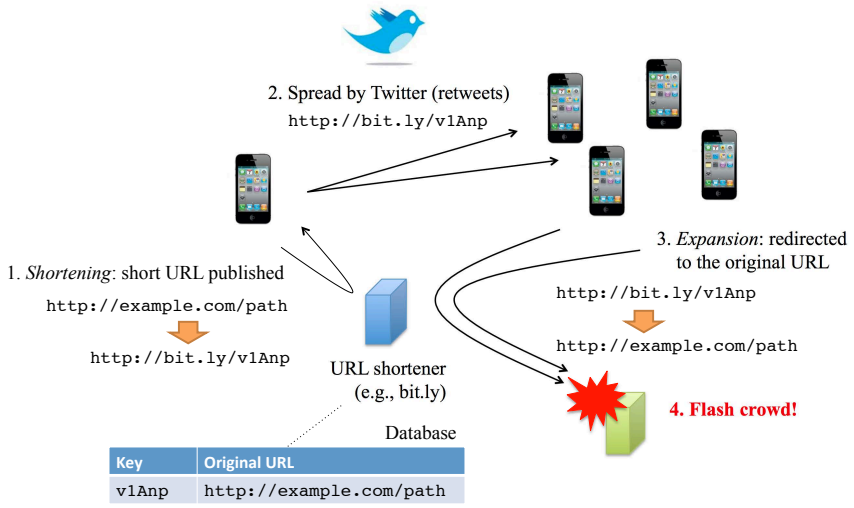

Figure 2: An example of shortening an original URL and expanding a short URL at a URL shortener.

many users jump on the site in a short period of time; this is a mechanism that causes a flash crowd.

The use of URL shorteners is transparent to Twitter users, because Twitter clients retrieve a short URL via APIs in the background and the original URL is automatically replaced with the short one. URL shorteners are usually open service; users are allowed to use them without prior registration and authorization. This openness can trigger malicious use.

2.3 CoralCDN

CoralCDN is a research network developed by Michael Freedman in 2004 [16]. It was designed to mitigate flash crowds; for example, CoralCDN distributed large quantities of amateur videos of the Indian Ocean tsunami in 2004. CoralCDN is fully open and requires no prior registration or authorization. It can be used just by adding `.nyud.net` to the domain name in the original URL, e.g., `http://example.com/path` → `http://example.com.nyud.net/path`. URLs including `.nyud.net` are called "Coralized URLs". CoralCDN can be used easily, but its use is not transparent to users since the replacement of URLs must be done manually.

CoralCDN is deployed on PlanetLab [13] and typically runs on 300–400 servers, spread over 100–200 sites worldwide [15]. It has sufficient capacity to handle 40–50 million requests per day. CoralCDN consists of DNS name servers and proxy servers; DNS servers maintain "A" records for `.nyud.net` and the proxy servers keep replicas of the original pages. Each replica is mapped to several proxy servers by Sloppy DHT technology [14], but the replica can be duplicated on other proxies depending on popularity. Replicas are updated after the expiry time, which is specified by the response header given in retrieving the original page. They are kept for at least five minutes, even if `No-Cache` directives are set in the response, and are removed within twenty-four hours at most.

3. SYSTEM OVERVIEW

This section overviews our URL shortener. We first show the usage in Section 3.1. Section 3.2 describes procedures of shortening and expanding.

3.1 Usage

We briefly describe the usage of our URL shortener here. The APIs shown in Table 1 are designed following bit.ly APIs [2].

Shortening original URL. We provide three ways to shorten a long URL.

- **API.** A Twitter client sends a URL to the "shorten API" in Table 1, and our URL shortener returns a short URL in a specified format. For example, issuing a request like `/api/shorten?longUrl=http://example.com/path&format=text`, and you will get a response of single line body like `http://rcdn.info/EXnXN5`.
- **HTML form.** A user submits a URL by the form on the top page (Fig. 3) [2], and the URL shortener then returns with a short URL.
- **Bookmarklet.** A user clicks the bookmarklet of

[2]We offer only Japanese pages, because very few people speak other languages in Japan and they are unlikely to cause flash crowds.

Table 1: APIs

Operation	URL
Shorten	`/api/shorten?longUrl={originalURL}&format={json|xml|text}`
Expand	`/api/expand?shortUrl={shortURL}&format={json|xml|text}`

Figure 3: The top page of our URL shortener; the title says "short URLs and CDN offer effective information sharing", the button says "(make) a short URL to CDN", and a short description accompanies the figure.

our URL shortener installed on the browser, which pops up a short URL of the current page.

Twitter clients that accept any URL shortener, such as TweetDeck [9] or YoruFukurou [12], provide transparency to users through the APIs, by setting our URL shortener as a default. The HTML form and the bookmarklet are offered to users of other clients.

Expanding short URL. We provide two ways to expand a short URL.

- **Redirection.** A user clicks a short URL, and our URL shortener then redirects her to CoralCDN.
- **API.** A Twitter client sends a short URL to the "expand API" in Table 1, and the URL shortener returns a Coralized URL without redirection.

3.2 Procedures of Shortening and Expansion

Figure 4 is an example of URL shortening and expansion at our URL shortener.

1. **Shortening original URL.** Upon receiving a request for shortening, the URL shortener then searches its database for the requested URL. If the URL is not found, the shortener generates a unique key for the URL by using the SHA1 hash function (the hash value is incremented if conflicted with existing keys), and stores a tuple of the unique key, the original URL, and the Coralized URL, into the database. Finally, the server returns the short URL that consists of the domain name and the unique key.
2. **Spreading URL.** Ditto.
3. **Expanding short URL.** Upon receiving a request for expansion, the URL shortener extracts the unique key from the short URL, and retrieves the corresponding tuple from the database. Our URL shortener then returns the Coralized URL to mitigate flash crowds.

4. DEVELOPMENT

This section describes our development story. In Section 4.1, we begin with what made us develop a new URL shortener though there are many existing shorteners. We describe development process of prototype and stable version of our URL shortener in Sections 4.2 and 4.3, respectively. Section 4.4 discusses our development strategy that worked well in the emergency.

4.1 Beginning

In March 2011, one of the authors, Takeru Inoue, lived in the Tokyo metropolitan area. He experienced the great quake there, but he was fortunately not in the terrible disaster area. On March 13th, he saw several tweets that said some web sites failed repeatedly, and then he suggested on Twitter to replace URLs with corresponding Coralized URLs. However, in the chaos following a disaster, we could not imagine that people will stay cool enough to replace URLs before posting messages. A mutual follower of his, Mr. Yuichi Yoshida or `@sonson_twit`, dropped an interesting tweet, "shouldn't URL shorteners translate original URLs into Coralized ones?" This tweet drove us to realize such a "Coralizing URL shortener". To realize the idea quickly, our first choice was not to develop it ourselves, but to ask existing URL shorteners, bit.ly and t.co, to rewrite URLs with some probability. We had no response unfortunately, maybe because it was weekend or they had no economic incentive. We also asked Akamai to provide a new URL shortener coupled with its great CDN, but no response again.

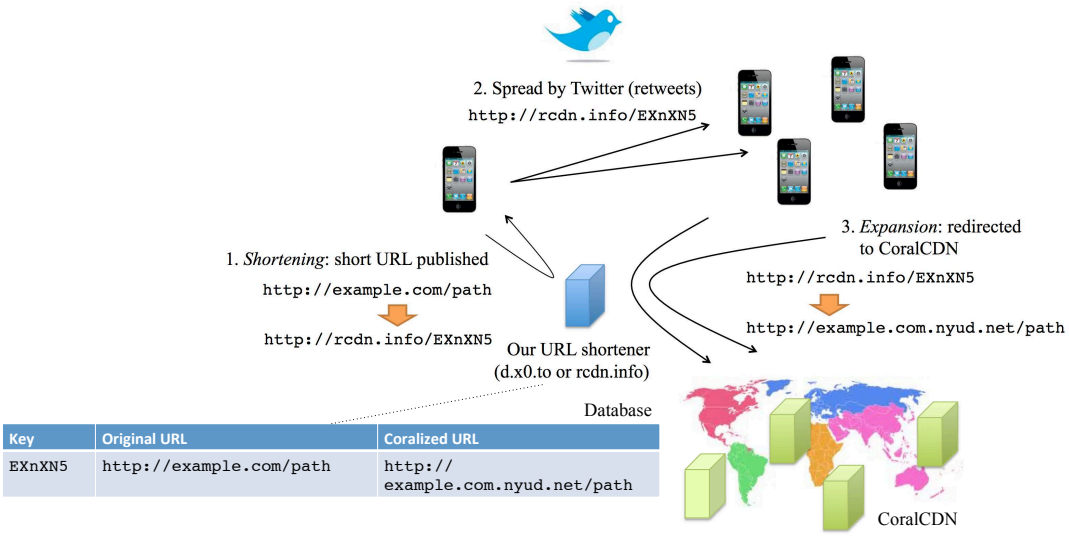

Figure 4: An example of shortening an original URL and expanding a short URL in our URL shortener.

4.2 http://d.x0.to

Finally, we decided to develop a new URL shortener. We implemented it on LAMP (Linux, Apache, MySQL, Perl) stack [17]. LAMP is a software bundle of free and open-source software. It comprises principal components to build a general-purpose web application server. Since common operations are built in the stack, developers are allowed to easily implement their own application servers.

We began the development around 13:00 on March 13th, and launched a prototype around 12:00 on March 14th[3]. The prototype required 541 lines of Perl script with some HTML pages. The top page was at `http://d.x0.to/top.html` (it is now pointing to `http://rcdn.info`). The prototype showed that it works pretty well to divert flash crowds. However, it had a performance issue; since we had no root privilege on a machine on which the shortener ran, the prototype was executed as a CGI program which had to be invoked for every request.

4.3 http://rcdn.info

In response to our call for a machine with the root privilege, some hosting companies gave us an offer. We chose Amazon Web Services (AWS) [1] because of the kind support. Our URL shortener is now running as a part of Apache server by mod_perl without heavy re-invocation. Despite lack of experience with AWS, we could install our system on the machine without major trouble; Linux ran on an "EC2" virtual machine (micro instance) in Tokyo region, and the domain name was maintained by a "Route 53" name server. Next, we considered a *cool* domain name for our URL shortener; the name should be short and be reminiscent of "redirecting to CDN". We named it "rcdn.info" and registered it with an ICANN accredited registrar [6].

We launched rcdn.info early on March 15th and added redirecting setting on d.x0.to. The launch was announced on Twitter around 10:00 on March 15th, "we've launched a new URL shortener, rcdn.info, for mitigating flash crowds against useful disaster pages". Our URL shortener appeared on several major online media like Yahoo! Japan [11] and livedoor [5][4] along with other disaster services supported by AWS; in the article, rcdn.info covered a large part with a figure.

4.4 Development Strategy

We implemented the URL shortener in an incremental manner. We first implemented just a few basic features, i.e., shortening and expansion. Following user requests, we also implemented APIs, bookmarklet, and several techniques to improve the latency; especially, APIs is a key feature to allow URL shorteners to be accessed transparently by Twitter client software. Our incremental approach worked very well, and so we could deploy it in a short time.

In spite of the rush job, we cared software quality so as not to confuse our users in an emergency. Moreover, the incremental development strategy helped us main-

[3]March 14th was a day off due to the disorder of public transportation.

[4]Yahoo! Japan has been the most popular site in Japan for more than ten years, and livedoor occupied the 8th Alexa ranking in Japan as of July, 2011.

tain the quality, since the features could be checked one by one. During the operation, just a single bug, which related to URL validation, was reported. It is worth noting that we implemented common security measures like SQL injection protection at the beginning.

We did not give priority to detecting malicious URLs, because CoralCDN maintains a global blacklist of specified domain names [15]. To improve security level, we are thinking of introducing Google Safe Browsing [4], which provides a list of suspected phishing and malware pages.

5. DATASET

This section describes the HTTP request log, which will be callsed our *dataset* hereafter, collected at our URL shortener. Section 5.1 overviews the dataset. Sections 5.2 and 5.3 examine the dataset and show users and topics of the shortener.

5.1 Overview

There were 24,959 HTTP requests at our URL shortener from March 15th 2:00 JST to 18th 22:23 JST[5]. Each log entry includes date and time, request method, path name, source IP address (and corresponding domain name), user agent, referring page, and so on.

To make the log publicly available, we cleansed the dataset to help analyses and carefully anonymized it for privacy preservation as follows. We removed requests other than expansion, e.g., requests for shortening, the top page, images, style sheets, and so on. We also removed requests from automated agents, whose user agent name includes "bot", "crawler", "slurp", or "AppEngine-Google", or whose domain name ends with "amazonaws.com". Source addresses are replaced with random integers. No referrer nor no user agent is included in the public dataset. Requests for only popular pages are included (a "popular page" means a page that had 5 or more expansion requests at our URL shortener). After all, the public dataset includes 3,791 requests; it is now publicly available online [7].

The launch announcement was made on the Twitter account of the author, Takeru Inoue or `@takemaru_jp`, who had 365 followers at that time. His followers mainly lived in the Tokyo metropolitan area, but the use of the URL shortener was not limited to the Tokyo area thanks to the major online media, as mentioned in Section 4.3. We confirmed that it was accessed from all prefectures in Japan.

In the following subsections, we see the public dataset (with some excluded attributes) from some aspects.

5.2 Users

[5]The log after March 18th was unfortunately lost due to mis-configuration.

Table 2: Referring pages

`Referer` header	Counts
No referrer	2,072
twitter.com	943
Twitter clients	149
SNS (e.g., facebook.com)	184
Others	443

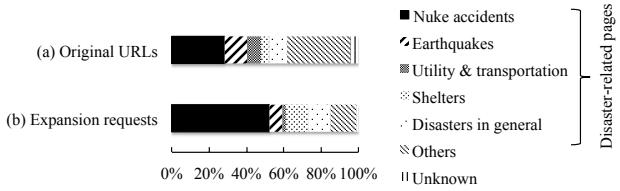

Figure 5: Topics of top 50 popular URLs at our URL shortener, and those of expansion requests for these popular URLs.

We analyze the `Referer` header, which is excluded from the public dataset, to identify our users. Table 2 shows the referring pages of the expansion requests. More than half of the requests have no referring page. We regard these "`Referer`-less" requests as issued by Twitter clients, since Twitter clients usually set no `Referer` header in the request. That is, 83.4 % of expansion requests were made by Twitter users. Social networking services (SNSs) accounted for a small part of the requests, because their users do not need to shorten long URLs. We can say that our URL shortener was mostly used by Twitter users.

5.3 Topics

During the logging period, 299 unique URLs were shortened. We manually categorize the top 50 popular URLs (pages) based on their topics in Fig. 5 (a). We also categorize the 3,582 expansion requests that were made for these top 50 URLs in Fig. 5 (b). Since these requests cover 94.4 % (3,582/3,791) of the dataset, we can understand user intents by examining just them.

We first discuss Fig. 5 (a). Our purpose in developing the URL shortener was to mitigate flash crowds for disaster information, but it was used in a variety of ways that differed from our original purpose. While 31 pages in the top 50 included disaster information, 17 pages were related to other topics. We could not retrieve valid content from the two URLs in the "unknown" category.

As shown in Fig. 5 (b), 85.1 % of requests that were made for the top 50 were issued for disaster-related pages. As expected, "nuclear accidents" garnered a high level of interest, more than half of the requests. This

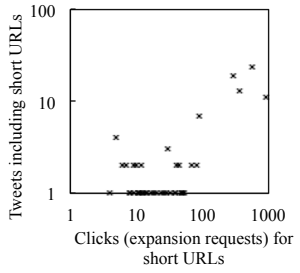

Figure 6: Correlation between tweets including rcdn.info URLs and clicks measured at the URL shortener.

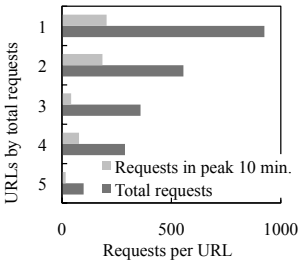

Figure 7: Total and peak requests for the five popular URLs.

chart shows that our URL shortener was mainly used to share disaster information.

6. ANALYSIS WITH TWEETS

This section examines our public dataset with tweets collected in the same period; the tweets were provided by another research project, as noted in Section 1. The tweets were collected as follows. We first created a list of experienced Twitter users who had 200 or more tweets on March 5th 2011, and then received their tweets from Twitter streaming API [8]. We had roughly 100 M tweets during the logging period (we have no means to estimate the sampling rate). We found 123 tweets that included short URLs published by rcdn.info.

Figure 6 shows the number of tweets that included rcdn.info URLs versus the number of clicks (expansion requests) for the URL. The figure shows a weak but positive correlation; the number of tweets is proportional to the number of requests raised to the power of 0.484, with the correlation coefficient of 0.691. We cannot determine the number of clicks per tweet, since the sampling rate is unknown. This positive correlation, however, implies that the click counts depends on tweets; more tweets yield more clicks.

In order to investigate flash crowds, we examine which tweets increased click counts sharply (Fig. 7). For the most popular URL, 22.1 % of clicks occurred within ten minutes after a "big" user tweeted (the user is `@DrTomabechi` who had 20 K followers). Similarly, for the second popular URL, 33.3 % of clicks were made within ten minutes of a tweet issued by another big user (`@kiwipon` who had 100 K followers). We also found such a big user for the fourth popular URL.

While the number of *total* clicks depends on the number of tweets (Fig. 6), click *rates*, especially flash crowd phenomenon, is strongly affected by big users (Fig. 7); big users have a key role in flash crowd mitigation.

7. CONCLUSIONS

This paper describes our experience with a new URL shortener coupled with CoralCDN in the aftermath of the Great East Japan Earthquake. The URL shortener redirects users to replicas on CoralCDN instead of original pages, in order to mitigate flash crowds. We began to develop it in response to frequent server failure and launched it just a few days after the earthquake. The interesting dataset was collected during the operation, and it was examined with tweets. Our dataset is now publicly available online for further collaboration to tackle future disasters.

Acknowledgement

We would like to thank Prof. Michael Freedman for developing an excellent content distribution network. We also wish to acknowledge energetic support in software development by Dr. Norihito Yasuda and Mr. Masaaki Nishino. We would like to thank Dr. Tatsuya Mori for his valuable advice in the log analysis. We would like to thank Mr. Yuichi Yoshida for designing the pages of the shortener. We wish to acknowledge kind support about Amazon Web Services by Dr. Yasuhiro Araki, Mrs. Miki Takata, and Mr. Keiichi Okabe. We would like to thank Mr. Genta Kaneyama for collecting massive tweets.

8. REFERENCES

[1] Amazon Web Services. http://aws.amazon.com/.
[2] bitly-api. http://code.google.com/p/bitly-api.
[3] Goo.gl challenges bit.ly as king of the short - NYTimes.com. http://bits.blogs.nytimes.com/2009/12/14/googl-challenges-bitly-as-king-of-the-short/.
[4] Google safe browsing API. http://code.google.com/intl/en/apis/safebrowsing/.
[5] livedoor news. http://news.livedoor.com/article/detail/5417888/ (in Japanese).
[6] Onamae.com. http://www.onamae.com/ (in Japanese).
[7] rcdn.info: Public dataset. http://rcdn.info/data.html.

[8] Streaming api — twitter developers. http://dev.twitter.com/docs/streaming-api.
[9] Tweetdeck. http://www.tweetdeck.com/.
[10] We want 'em shorter. — MetaFilter. http://www.metafilter.com/8916/.
[11] Yahoo! news. http://headlines.yahoo.co.jp/hl?a=20110316-00000003-rbb-sci (in Japanese).
[12] Yorufukurou. http://sites.google.com/site/yorufukurou/home-en.
[13] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33:3–12, 2003.
[14] M. Freedman and D. Mazières. Sloppy hashing and self-organizing clusters. In *Peer-to-Peer Systems II, LNCS*, volume 2735, pages 45–55. Springer, 2003.
[15] M. J. Freedman. Experiences with CoralCDN: a five-year operational view. In *USENIX NSDI*, 2010.
[16] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *USENIX NSDI*, 2004.
[17] J. Lee and B. Ware. *Open source Web development with LAMP: using Linux, Apache, MySQL, Perl, and PHP*. Addison-Wesley, Dec. 2002.
[18] T. Sakaki, F. Toriumi, and Y. Matsuo. Correlativity of network traffic and social media for a massive earthquake. In *ACM CoNext Special Workshop on the Internet and Disasters*, 2011.