WebCloud: Recruiting social network users to assist in content distribution

Fangfei Zhou† Liang Zhang† Eric Franco† Alan Mislove† Richard Revis‡ Ravi Sundaram†
†*Northeastern University* ‡*Jandrell, Pearson & Revis Ltd.*

***Abstract*—Today, the data exchanged over online social networks (OSNs) represents a significant fraction of Internet traffic. However, OSN content is different from more traditional web content, as it is more likely to be generated at the edge of the network, to be exchanged within a local geographic region, and to possess a more even popularity distribution with fewer popular objects. Unfortunately, most OSNs still use largely centralized approaches to distribute content (e.g., CDNs and web caches), resulting in lower performance due to the different workload.**

In this paper, we take a first step towards addressing this situation by proposing WebCloud, a content distribution system for OSNs that works by repurposing client web browsers to help serve content to others. When a user browses content, WebCloud tries to serve the request from one of that user's friends' browsers, instead of from the OSN directly. Unlike other systems, WebCloud works with existing browsers and does not require any plug-ins, and therefore can be directly applied to today's OSNs. We demonstrate the practicality of WebCloud with microbenchmarks, simulations of a Facebook deployment, a real-world deployment, and evaluations of a proof-of-concept iOS app.

I. Introduction

With online social network (OSN) traffic making up a significant fraction of Internet traffic [1], [2], OSNs have changed the way we use the Internet. Today, massive amounts of content are being created and shared at the edge of the network via OSNs. However, existing web content distribution architectures—built to serve more traditional workloads—are ill-suited for these new patterns of content creation and exchange. Web caches and content distribution networks (CDNs) have been shown to exhibit poorer performance on OSN content [3], [4], causing many OSNs to move from CDNs to highly-engineered in-house solutions [5]–[7].

Given the workloads present on OSNs, allowing end users to assist in content distribution to others would be a natural fit. In fact, a number of peer-to-peer (p2p) systems for implementing OSNs or distributing content have been proposed, including FireCoral [8], Akamai's NetSession [9], PeerSoN [10] and Diaspora* [11]. However, these systems either (a) are built as separate (non-web) systems or (b) require plug-ins, both of which drastically limit applicability and userbase.[1] Other approaches include decentralizing the OSN's data center architecture [15] into many regional data centers, but require significant changes and expense for the OSN. Thus, in order to serve as a practical alternative for content distribution for today's OSNs, any new approach would need to work via the web and not require any changes (e.g., plug-ins) by end users.

In this paper, we take a step towards addressing this situation by introducing WebCloud, a content distribution system designed to work with existing OSNs. WebCloud operates via the web, does not require any plug-ins, and works by recruiting users' web browsers to help serve content to other users. We demonstrate that by deploying WebCloud, OSNs would enjoy most of the benefits of large centralized CDNs with dramatically lower costs.

Our contributions in this paper are as follows:

- We examine real-world OSN data and demonstrate why existing content delivery systems do not work well when applied to OSN content.
- We present the design of WebCloud, which is based on JavaScript added to the OSN pages, coupled with middleboxes called *redirector proxies*.
- We evaluate WebCloud using microbenchmarks, a small-scale deployment, large-scale Facebook simulations, and proof-of-concept iOS app.

The rest of this paper is organized as follows. Section II explores the changing workloads and discusses the implications of these findings, motivating the design of WebCloud. Section III details the design and implementation of WebCloud, and Section IV presents a detailed evaluation of WebCloud. Section V discusses related work and Section VI concludes.

II. Patterns of content exchange

We now take a closer look at the trends that serve as our motivation, using real-world OSN content-sharing data.

A. Data set

Our data set was collected on December 29, 2008 by crawling New Orleans Facebook regional network [17]. We

[1]Even the most popular [12] of all FireFox plug-ins (Adblock Plus) is installed by only 4.2% of users (AdBlock Plus reports approximately 15 million active users [13], out of approximately 350 million Firefox users [14]).

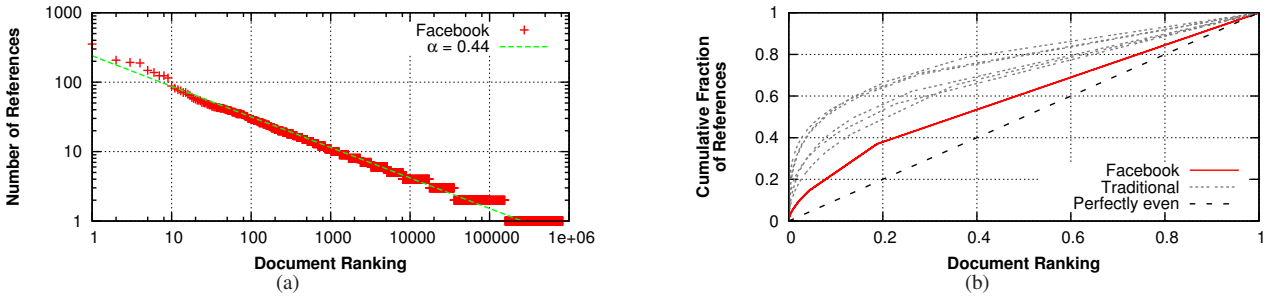

Figure 1. Graphs showing (a) the distribution of photo popularity, matching a Zipf distribution with $\alpha = 0.44$, and (b) the cumulative distribution of photo views compared to five different traditional web workloads [16] and a perfectly even distribution. A significantly lower scaling exponent is observed, leading to a popularity distribution with fewer "hits" and a fatter tail.

conducted a breadth-first-search of all New Orleans network users, in the same manner as in previous work [18]. By default at the time of the crawl, Facebook allowed all users in the same regional network to view each others' profiles, and we were thus able to crawl a large portion of the New Orleans network. In total, we collected information on 63,731 visible users connected together by 1,545,686 undirected links.

As we are interested in content exchange, we also collected information about the photos that users exchange. Because data on photo *views* is not available, we use photo comments as a proxy for views[2] (i.e., if a user has commented on a photo, they must have viewed it). Crawling the news feed in a manner similar to previous work [20], we discovered information on a total of 1,068,787 comments placed on 816,508 photos.

B. Properties

We now explore a few of the properties of this content exchange workload.

Content is created at the edge We first explore *where* the emerging content being exchanged over the web is being created. Today, the rapid adoption of smartphones, digital cameras, digital camcorders, and professional-quality music and video production software, combined with the low cost of broadband Internet service, has greatly eased content creation by individual users. Significantly more news articles are written by bloggers than news organizations [21], more photos are shared on online social networks [22] than on professional photography websites [23], and much of the content shared on YouTube, the most popular video-sharing site, is created by end users [24] empowered by the ubiquity of smartphones and webcams. The net result is that a significant fraction of Internet traffic contains content that is created at the edge of the network [1], [2].

[2]Recent work [19] has shown that while visible interactions do not perfectly capture latent interactions, the two share a number of statistical properties and, in particular, are much more similar to each other than to the properties of the social network alone.

Content is of more uniform popularity We now explore the *popularity distribution* of the content in emerging workloads, relative to previous workloads. To do so, we examine the popularity of photos on Facebook, and then compare the popularity distribution to that observed in studies of traditional web workloads [16]. The results are presented in Figure 1. We first note that, like traditional workloads, the Facebook request pattern follows a Zipf distribution, with an exponent of $\alpha = 0.44$.[3] However, there is one primary distinction with respect to traditional workloads: The Facebook workload contains a significantly lower exponent of the Zipf distribution (for reference, the exponents of the traditional web workloads range from 0.64 to 0.83 [16]). Thus, the newly emerging workloads have less emphasis on popular items, resulting in a more uniform popularity distribution and a significantly longer, fatter tail [25].

Exchange is governed by the social network We turn to explore *how* users are locating content. We calculate the fraction of comments on photos that come from the local social network of the uploader. The result of this analysis is that over 28.3% of the comments are placed by friends of the uploader, and at least[4] 89.1% are placed by friends or friends-of-friends (compared to expected values of 0.04% and 0.30%, respectively, were the placement random). This indicates that users are significantly more interested in the content that is uploaded by their friends and friends-of-friends. We have also found similar trends in browsing behavior in other online social networks: 80% of the photos viewed in Flickr were found by browsing the social network [18].

Exchange has significant geographic locality Finally, we explore the connection between content exchange and *geographic locality*. Using our Facebook data set, we find that

[3]The exponent of the Zipf distribution was calculated using a maximum-likelihood estimator. We observed a log-likelihood of -2602, strongly supporting a Zipf distribution.

[4]Our estimate of the fraction of comments placed by friends-of-friends is actually a lower bound: It is possible that two New Orleans users who are not friends or friends-of-friends within the New Orleans network are friends-of-friends when considering non-New Orleans users.

32.9% of the friends of New Orleans users are also in the New Orleans network[5]; similar findings have been observed in other regional networks [20]. However, if we examine the fraction of content exchange that occurs between New Orleans network users, we observe that 51.3% of comments are placed by other users within the New Orleans regional network. This indicates that the significant geographic locality already present in social networks is present to an even greater degree in the content exchange that occurs over these networks. A similar conclusion was found in [15], where the authors found that traffic local to a region is produced and consumed mostly in the same region.

C. Discussion

The content that is increasingly being shared on the web today is created at the edge of the network, but is exchanged using centralized infrastructure. The usefulness of existing techniques on this workload is declining [3], [4], [15]: For example, caching the most popular 10% of the items in traditional workloads would satisfy between 55% [16] and 95% [26] of the requests; in our social network workload from the previous section, such a cache would only satisfy 27% of the requests. This also affects the ability to use CDNs, which similarly work best for popular content. As the amount and size of end-user-generated content increases, this centralized approach is likely to become a bottleneck, limiting the ability for users to exchange new and larger content.

The most natural approach to address the changing workload is to work towards more decentralized content exchange. While some have suggested decentralizing the OSN's data center architecture [15] into many regional data centers, this requires significant changes and expense for the OSN. Instead, we focus on retaining the centralized OSN architecture of today, while attempting to decentralize content exchange when possible. In fact, a number of peer-to-peer content exchange applications exist (including decentralized social networks like PeerSoN [10] and Diaspora* [11]), but their techniques are not applicable to web-based content exchange. In order to serve as a drop-in replacement for current distribution architectures, any new approach would need to work using web technologies.

III. WebCloud design

In this section, we describe the design of WebCloud, a system that takes the first steps towards decentralized content exchange on existing social networks.

A. Overview

We begin by describing the deployment scenario that we expect for WebCloud. First, WebCloud is designed to be deployed by a web site, such as the provider of an online social network. We shall refer to the operator of this site as the *OSN* for the remainder of this section. Second, WebCloud is designed to be compatible with the web browsers of today. Third, WebCloud is designed to serve as a cache for content shared between users, much in the same manner as CDNs today serve as a cache for popular content. Thus, should WebCloud not be able to serve a particular request, we assume that the OSN has a copy of the content located on their servers, and can serve the request.

B. Keeping content exchange local

The web operates in a client–server manner: without plug-ins, it is not possible to have one web browser fetch content directly from another. Thus, we first examine how closely we need to approximate direct communication between web browsers to improve content distribution.

As observed in section II-B, content exchange in online social networks has significant geographic locality; if we can keep the content exchange between two users within the same ISP and geographic region, we can address many of the concerns in Section II-C. In more detail, suppose that the two users who are exchanging content are served by the same ISP. In this case, we argue that keeping the content exchange within the ISP, even if not directly between the users' browsers, is sufficient to reduce both the OSN's and the ISP's costs. For the OSN, keeping content exchange within the ISP obviates the need for the OSN to serve the content, reducing the cost of bandwidth and serving infrastructure. For the ISP, keeping the content within its network removes the bandwidth cost of transit via another ISP. These observations are similar to those in the recent work on keeping peer-to-peer traffic local [27], as well as those used to place CDN servers [28].

C. Design

At a high level, WebCloud emulates direct browser-to-browser communication by introducing middleboxes called *redirector proxies*. These proxies—located within each geographic region of ISPs similar to CDN servers—serve as a relay between web browsers. The proxy determines if any other online local user has the requested content, and if so, fetches the content from that user's browser and transmits it to the requestor. Should no local user have the content, the browser fetches the content from the OSN.

Figure 2 gives an overview of the design of WebCloud; below we describe the details.

1) WebCloud content: WebCloud is designed to be a drop-in component that can be deployed by existing web sites. WebCloud is agnostic to the type of content that is exchanged; however, WebCloud requires that content be named using content hashes (i.e., the name of a piece of content is its hash). This is necessary in order to ensure that content cannot be forged, and is discussed in more detail in Section III-D.

[5]The actual fraction of users who live in New Orleans is likely even higher, as we only consider users who explicitly joined the New Orleans network.

Figure 2. Diagram comparing approaches to content sharing. (a) Content sharing on existing online social networks, where Alice first uploads content to the provider and then Bob requests it. (b) Content sharing in WebCloud, where Alice first informs the proxy of locally stored content. When Bob requests content from the proxy, the proxy fetches it from Alice and delivers it to Bob, thereby keeping the content exchange local.

2) WebCloud client: To deploy WebCloud, the OSN includes the WebCloud JavaScript library in their web pages. This JavaScript serves two functions: communicating with the proxy, and storing and serving local content.

To communicate with the proxy, the WebCloud JavaScript opens and maintains an active connection to the local redirector proxy.[6] This connection is based on XMLHTTPRequests (XHRs) using long polling, or WebSockets.[7] Using XHRs, the client always maintains an outstanding, unanswered XHR to the proxy (should this request time out, another is simply created). This allows the proxy to send messages to the client (by finally responding to this request with the content of the message), and the client to send messages to the proxy (by creating a new XHR with the message). Using WebSockets, the client uses a single bi-directional communication channel to the proxy.

To store and serve local content, the WebCloud JavaScript uses the LocalStorage API [29] that is supported in most modern web browsers. In brief, LocalStorage allows a web site to store persistent data on the user's disk. LocalStorage is similar to cookie storage, but it is larger in size and can be programmatically accessed via JavaScript more easily. When a user views content in WebCloud, the JavaScript stores this content in the user's LocalStorage, treating it as a least-recently-used cache.

Finally, in order to take advantage of WebCloud, the OSN must load content using WebCloud. To do so, the WebCloud JavaScript code exports an API that the OSN can use:

- `connect()` Called when the web page is first loaded. Causes the WebCloud library to connect to the nearest redirector proxy and establish an open session.
- `load(content_hash, id)` Called when the client code wishes to load an object. Causes WebCloud to request the object with the corresponding content-hash from the proxy. If no peer has the content, the JavaScript code loads the content from the original web site. The `id` refers to the DOM id of the object; WebCloud will display the object once it has been loaded.

3) Redirector proxy: Redirector proxies allow clients to fetch content from each other. Each redirector proxy maintains connections to all of the OSN's online clients that are located within the same geographic region and ISP. This list is kept up-to-date as clients join and leave.

The proxy keeps a list of the content that each client stores in LocalStorage. For example, as shown in Figure 3, Alice requested content `AB4536A`, the proxy would fetch the content from Charlie and then deliver it to Alice. Should a proxy be unable to serve a request using the local clients or its local cache (or if fetching the content from Charlie fails), the proxy returns a `null` response to the client, who then fetches the content from the OSN's servers in the same manner as today. Thus, the performance of WebCloud (measured by the fraction of requests served by a connected client) is heavily dependent on the browsing patterns of users and their online/offline behavior. We demonstrate in Section IV that these patterns on today's sites are amenable to WebCloud, garnering a significant hit-rate.

An optional local cache can be included in the redirector proxy to help content exchange. Such a cache would be present if, for example, an existing CDN deployed WebCloud in conjunction with the already-deployed edge servers.

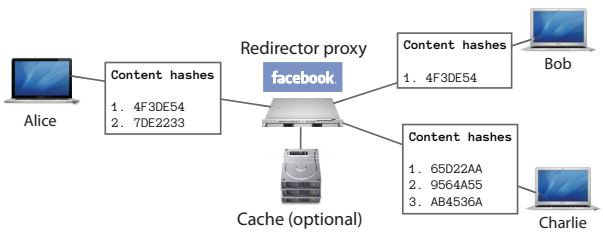

Figure 3. Diagram showing the state the redirector proxy keeps for each client, consisting of a list of the content ids each client is storing locally.

[6]The local redirector proxy can be located using DNS techniques similar to those used by CDNs to locate the nearest CDN server.

[7]WebSockets is a newly developed standard that is seeing acceptance in major browsers. As of this writing, it is available in the newest versions of Safari, Firefox, and Chrome.

4) Communication protocol: The protocol between the WebCloud JavaScript running at the client and the redirector proxy consists of three message types, all encoded on the wire using JSON, described below:

- `update({content_hash, ...})` The client sends an `update` message to the proxy to inform the proxy of the set of locally stored content. The message contains the list of content hashes that the client has in its LocalStorage, and the proxy records this list.
- `fetch(content_hash)` A `fetch` message can be sent either from the client to the proxy or vice versa, and contains the content hash of the content requested. When the client sends the fetch message to the proxy, the proxy checks to see if any other client has reported having that content in its LocalStorage. If so, the proxy forwards the fetch message to that client. If not, the proxy returns a `null` response to the requestor.
- `response(content_hash, content)` This message is a reply to the `fetch` message, and contains the content hash and the content itself.

A typical session would consist of a client connecting to the proxy and sending an `update` message to inform the proxy of the content that is locally stored. The client and proxy then exchange `fetch`, `response`, and `update` messages, as both the local user and other remote users browse content, and the client finally leaves by closing the connection.

D. Security

We now examine how WebCloud handles malicious users. There are three primary concerns: malicious users might attempt to serve bogus content to other users; malicious users may try to view content they are not allowed to; and malicious users might attempt to perform a denial-of-service attack by overloading the proxy or violating the WebCloud protocol. First, in order to detect bogus content, as discussed above, all content in WebCloud is identified by its content hash, so the proxy and all users are able to immediately detect and purge forged content. Second, using content hashes for naming also enables WebCloud to authenticate content requests, as users can only request content if they know its hash.[8] Third, in order to address users who attempt denial-of-service attacks by violating the protocol, WebCloud uses similar techniques that are in-use by such sites today: OSNs such as Facebook often block accounts, IP address, or subnets where malicious behavior is observed [30]. Since the redirector proxy is under the control of the OSN, existing defenses against these attacks can be deployed at the proxy as well.

[8]This is precisely the semantics that is followed by Facebook and other sites today; the URLs of images are obfuscated, but anyone who possesses the URL can download it.

E. Privacy

Next, we examine whether WebCloud changes the privacy implications of sharing content. First, WebCloud only allows users to fetch content that they could access anyway (see Section III-D), so users cannot abuse WebCloud to view content they otherwise could not. As a result, WebCloud does not allow users to disclose content to unauthorized third parties who they could not already.

Second, in WebCloud, users do receive some information about *views* of content by other users. However, due to the indirect nature of the communication, users are unable to determine *who* is viewing the content. Thus, WebCloud effectively provides k-anonymity [31] to browsing users, where k is the number of online users connected to the same proxy who are able to view the photo. Many OSNs already provide some form of view information to users (e.g., view counts, or names left beside comments on content), so WebCloud often does not leak information that was not already available.

Regardless, there may be corner cases where k-anonymity is insufficient (e.g., if a user only makes a piece of content visible to a single other user, effectively removing viewer anonymity). In such cases, the OSN can disable loading such content via WebCloud (or can allow the browsing user to do so). Alternatively, the OSN can configure the proxies to add background requests to random pieces of content, sometimes referred to as *cover traffic* [32], in order to obfuscate requests.

F. Mobile users

The description of WebCloud so far has focused on users who are connected from traditional web browsers. However, users are increasingly accessing services like OSNs from mobile devices. Below, we address the technical issues associated with deploying WebCloud on mobile devices and discuss the unique challenges that they present.

As the mobile web browsers on both iOS and Android support both LocalStorage and XHRs, WebCloud works without modification on these devices when the user visits the OSN's web site. However, a potential drawback is that the session times are likely to be much shorter, as smartphones typically only allow users to have one "active" web page at a time (thus, whenever the user browses away from the site, closes the browser, or puts the phone to sleep, the connection to the redirector proxy will be broken). Luckily, the popularity of site-specific apps (e.g., Facebook for iOS) enables an alternative: WebCloud can be integrated into the app, and use the background service support provided by both platforms to maintain a connection to the redirector proxy.

Smartphones present two additional unique constraints: Limited battery life and data access charges. The background service support present on many mobile operating systems holds the potential to help with the former, as the app itself is

Table I
AVERAGE TIME TO LOAD FACEBOOK PHOTOS.

Accessed from	Served from		
	Facebook	WC-LAN	WC-Cable
LAN	668 ms	63 ms	398 ms
Cable	690 ms	153 ms	532 ms

not required to be running constantly. Regardless, the impact of WebCloud on battery life is still unclear. In order to gauge this impact, we implemented a prototype WebCloud app on iOS. Fully described in Section IV-D, we demonstrate that the impact on battery life and data usage is acceptable.

IV. EVALUATION

We now present an evaluation of WebCloud. We are guided by four questions:

- First, what is the complexity of WebCloud, and how does it perform in practice? In other words, is there any discernible difference for the end users?
- Second, how would WebCloud perform if a large-scale provider were to deploy it? What fraction of content exchange could be served via WebCloud?
- Third, how does WebCloud perform when deployed on mobile devices? Is battery life significantly impacted?
- Fourth, how does WebCloud perform when deployed on a real social networking site and used by real-world users?

A. WebCloud implementation

We implemented WebCloud to work in conjunction with Facebook's photo-sharing service. Photos are one of the most popular content-exchange mechanisms on Facebook, allowing us to easily obtain a userbase and a workload. The prototype redirector proxy is implemented using Python, most of which is the low-level communication support code for WebSockets and XHRs. The client-side support is implemented using JavaScript.

In order to deploy our WebCloud prototype, we set up a web proxy that was configured to inject the WebCloud JavaScript when serving Facebook's JavaScript files. Thus, users installed WebCloud by configuring their browser to fetch content via this web proxy. Additionally, the proxy modified Facebook's photo loading code so that all content requests for photos were served by WebCloud, thereby allowing WebCloud to work with optimizations like photo pre-fetching. From an end user's perspective, it appeared as if Facebook had deployed WebCloud.

B. Microbenchmarks

We begin our evaluation with a few microbenchmarks.

1) Content loading latency: We first examine the latency incurred in downloading photos. Since loading photos using WebCloud requires the request to be routed via a redirector proxy and served by the browser of a remote client, we explore whether any additional latency is incurred. To do so, we uploaded a set of 10 new Facebook photos (average size 62 KB) and then downloaded them in one of three ways: from Facebook as normal, from a WebCloud client connected to the same LAN as the redirector proxy, and from a WebCloud client connected via a cable modem. We ran each of the above three tests in two ways: downloading the photo to a machine on the same LAN as the redirector proxy, and downloading the photo to a machine using a cable modem. The proxy and all of the clients are located in the city of Boston. Note that we uploaded new photos for each experiment, in order to avoid any effects from in-network caching. For each configuration, we report the average download time across 10 experiments.

The results of this experiment are presented in Table I. First, in all cases, loading the content using WebCloud was actually *faster* than loading it directly from Facebook. This results from running our experiment all on machines in Boston; loading data from Facebook requires downloading it from California. As we expect a redirector proxy to be placed in each ISP region, this result is representative of what would be expected in practice. Second, accessing content that is stored on a client connected via a typical home cable modem added approximately 350 ms of additional latency in both tests, due to the slower connection. However, in all cases, the latency of WebCloud is acceptable, especially when used in conjunction with Facebook's photo pre-fetching.

2) Storage size: Second, we examine the number of photos that can be stored in each user's LocalStorage. We collected a sample of 954 Facebook photos from New Orleans users, and found that the sizes ranged from 12 KB to 226 KB, with a median of 67 KB. By default, Chrome, Safari, and Firefox all set a maximum size of 5 MB for the LocalStorage per (domain name, port) pair. Taking into account the 33% overhead induced by storing photos in base64 format, WebCloud is able to store 56 photos, on average, in each user's LocalStorage.

As the 5 MB storage limit is per domain and port, WebCloud extends the storage limit arbitrarily by loading stub JavaScript from multiple domain names (e.g., `foo1.facebook.com`, `foo2.facebook.com`, ...) and ports, all pointing to the same redirector proxy. By sending messages via `postMessage` in JavaScript, WebCloud ensures that these stub scripts can communicate, allowing the multiple LocalStorage instances to all be accessible. Regardless, as we demonstrate below, even using only 5 MB allows most of the savings of WebCloud to be realized.

3) Redirector proxy scalability: We are interested in understanding the rate of `fetch` requests that a single

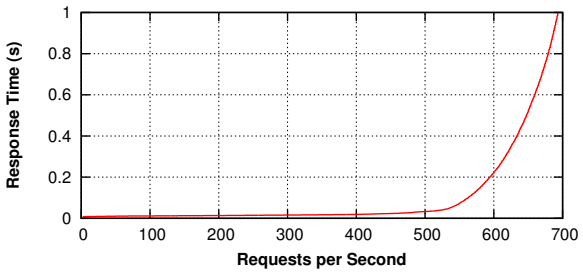

Figure 4. Average response time versus request rate for the WebCloud redirector proxy. The proxy can support over 500 requests per second before incurring significant additional latency.

proxy can handle, as this serves as the dominating factor controlling the number of online users that the proxy can support. For this experiment, we ran our redirector proxy implementation on an 4-core 2.83 GHz machine with 16 GB of memory. We then connected an increasing number of clients to the proxy, all located on the same LAN, where each client was configured to continually issue content `fetch` requests for a 85 KB photo every five seconds. Finally, we examined the tradeoff between the rate of incoming requests and the response time (measured as the time elapsed from the start of the request until the last byte of the content arrives) at the clients.

The results are presented in Figure 4 and show that our prototype redirector proxy implementation is able to support over 500 content `fetch` requests per second before more than 10 ms of latency is incurred. If we assume that on average user issues one content request per minute, this represents a single redirector proxy supporting over 30,000 online users. It is also worth noting that a highly-optimized implementation of our proxy is likely to support a much greater number of users.

C. Simulation

Next, we evaluate the potential of WebCloud at scale by simulating Facebook deploying WebCloud in one region.

1) Generating traces: For the simulation, we need a number of pieces of input data: a social network, a trace of when users are online and offline, and trace of when users browse each other's photos. Unfortunately, a detailed trace of Facebook user online/offline and photo viewing behavior is not widely available. Instead, we use our Facebook data discussed in Section II-A to generate synthetic traces. The trace is generated so as to preserve the bias that is present in user online/offline behavior, photo uploads per user, and photo views per user. Below we detail our methodology for generating one-week synthetic traces in the WebCloud evaluation, and compare the traces to studies of real-world systems.

Photo uploads To generate a photo upload event, we need to select two things: the uploading user and the time of the upload. To select the uploading user, we choose randomly from the list of uploaders observed in the photo comments data. Users who were observed to upload more photos are more likely to be chosen. This method preserves the non-uniform distribution of photo uploads across users. Then, we randomly select an upload time during the simulated week.

Photo views To generate a photo view event, we need to select three things: the viewing user, the time of the view, and the photo that is being viewed. First, we use a similar mechanism as above, selecting the viewing user randomly from the list of users who placed comments (again, this preserves the fact that certain users view photos more than others). Second, to select the time of the view, we pick a day of the week and time of the day randomly from the list of timestamps of the viewing user's comments. We add a small amount of random time (between -30 and 30 minutes) to this timestamp to ensure that many views do not happen at once. Selecting the time in this way preserves the daily and weekly trends in browsing behavior.

Third, to select the photo that the user views, we first select an *uploading user*, one of whose photos the viewing user will view. The uploading user is selected randomly from the list of users whose photos the viewing user commented on. This method preserves the fact that users are more likely to view certain other users' photos. We then select which of the uploading user's photos is viewed using a weighted distribution (since more recent photos are more likely to be viewed). The weights are derived from a study [33] of the views received by Flickr photos. In brief, photos were observed to receive 37.2% of their views on the first day, 21.3% on the second day, and so forth. Thus, all photos uploaded by the uploading user in the previous day have an even likelihood of being chosen, which is higher than all photos uploaded two days ago, etc.

Online/offline trace Finally, we generate an online/offline trace for the users based on the photo view trace just described. For each user, we look at the timestamps of their photo views. For each timestamp T, we simulate the user coming online at time $T - v_-$ and simulate the user going offline at time $T + v_+$, with v_- and v_+ selected randomly between 1 and 30 minutes. The result is that, for each view, the user is online for between 2 and 60 minutes (with an average of 31 minutes that approximates recent session time studies [34]).

Evaluating traces We now briefly compare the synthetically generated trace to empirically gathered data regarding use of Facebook and other social media sites. First, we examine the average time spent per user online during the week. Across all traces, the average time online per week ranged from 1.77 to 6.02 hours, which matches favorably with studies of real-world Facebook usage [35]. Second, we examine the number of online users during the course of our simulated week and observe the strong diurnal patterns that would be expected in realistic workloads. Third, we examine the pop-

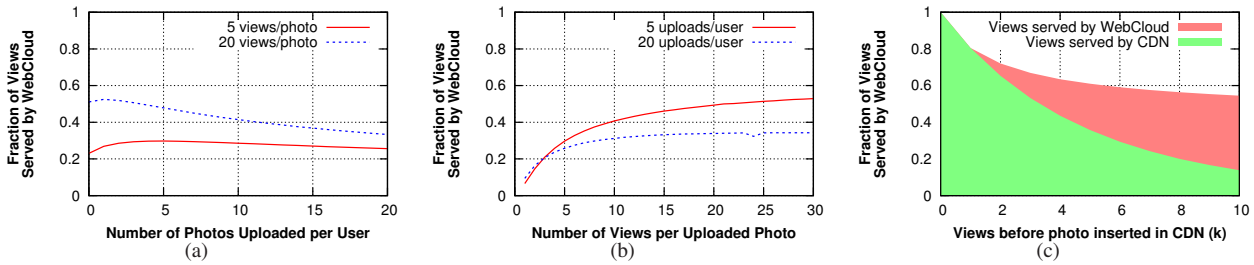

Figure 5. Fraction of views served by WebCloud in various simulations. (a) WebCloud hit-rate as the average number of photos uploaded per user is varied. (b) WebCloud hit-rate as the average number of views per photo is varied. (c) WebCloud hit-rate when run in conjunction with a CDN, varying the minimum number of hits (k) before a photo is placed in the CDN.

ularity of individual photos themselves. The overall photo popularity distribution closely matches previously observed distributions of photo popularity in social networks [36].

2) Simulation results: There are two primary questions that we are interested in: First, what is the hit-rate of WebCloud? In other words, what fraction of the photo views can be served from another online user via WebCloud, instead of directly from the OSN? Second, how does WebCloud compare to a CDN? In other words, if a CDN is used in conjunction with WebCloud, what additional hit rate does WebCloud provide?

For these experiments, we only consider views that are not served from the user's local cache. We simulate all clients as being able to store 50 photos in their LocalStorage. To explore the various environments that WebCloud may be deployed in, we vary two parameters: the number of photo uploads and the number photo views.

For clarity, we express the average number of photos uploaded *per user*, and the average number of views *per photo*. For all experiments in this section, we repeat the experiment 10 times with different random seeds and report the average.

Varying the number of uploaded photos We begin by examining how the number of uploaded photos affects the hit rate of WebCloud. We run simulations with two different settings of the average number of views per photo (5 and 20). As the average number of photos uploaded per user increases, we expect that the WebCloud hit-rate will initially increase as the collective LocalStorage fills. However, the hit-rate will then begin to drop off, once each user's LocalStorage fills up.

The result of this experiment is presented in Figure 5(a). We see that for each setting, the hit-rate has a maximal peak, matching our intuition from above. However, we note two trends: First, the fall-off in hit-rate is rather slow, falling only by about 20% as the number of photos uploaded increases from 5 per user to 20 per user. Second, the overall hit-rate is surprisingly high, ranging between 23% and 57%. The upshot is that even in this wide variety of configurations, WebCloud serves a significant fraction of the views.

Varying the number of views per photo Next, we turn to evaluate how the average number of views per photo affects WebCloud's performance. As the the number of views increases, we expect that the hit-rate of WebCloud to increase, due to the bias towards viewing recently uploaded photos. Similar to the previous experiment, we present results from two different configurations of the average number of uploads per user (5 and 20). The result of this experiment is presented in Figure 5(b). We observe that our expectation holds: as the average number of views per photo increases, the hit rate of WebCloud rises.

Comparison to a CDN To answer our final question, we simulate WebCloud running in conjunction with a CDN. In these simulations, clients first query the CDN; only if the CDN does not have the content, the request is forwarded to WebCloud. However, it is unrealistic to assume that *every* photo would be inserted into the CDN, as sites today only use such services to serve popular content. Thus, we configured the CDN to only store content that been requested k times. Increasing k implies that only more popular content is served via the CDN. We assume that the CDN has unlimited storage, and all photos have zero views at the beginning of the experiment.

The results of this experiment are shown in Figure 5(c). For example, if $k = 0$, the CDN serves all content, forwarding no requests to WebCloud. However, if k is increased to 5, WebCloud serves over 25% of the requests, and at $k = 10$, WebCloud serves over 40%. This result shows that even if the OSN uses a CDN, WebCloud still provides a significant hit rate.

D. Mobile devices

We now turn to examine WebCloud on mobile devices. Recall from our discussion in Section III-F that we are concerned with two questions: First, can WebCloud be feasibly implemented as a background service? Second, if so, what is the impact on battery life and data usage? To evaluate this, we implemented a prototype WebCloud app on iOS. Our app registers itself as a background VoIP service, allowing it to maintain a persistent connection with

the redirector proxy even if the app is not in the "active" application.

To evaluate the impact on battery life, we deployed our WebCloud app to an iPhone 4 running iOS 4.2 configured to connect to a test redirector proxy. The proxy issued `fetch` requests for the app to serve a 60 KB photo every five seconds, allowing us to measure the number of photo requests that could be served from the phone before running out of battery. We found that the WebCloud app could serve 5,031 requests over 8.26 hours when connected via 3G, and 24,700 requests over 34.9 hours when connected via WiFi. Given the fact that, even in the most demanding of the simulations in Section IV-C2, the user with the *heaviest* workload served 160 photos per day (and the average user served 4.36 photos per day), the WebCloud app is likely to only consume a small amount of the battery life.

The data from our simulations also addresses the concerns over data usage. Even under the heaviest workload, the most-loaded user served a total of 72 MB during the simulated week, while the average user served 2 MB. Both are within the data allocation provided by most 3G providers.

E. Deployment

As a final point of evaluation, we deployed our WebCloud prototype on a small scale within our department at Northeastern University to examine how it would work with real-world users.[9] We recruited users by emailing our graduate students.

Over the course of our 10-day deployment, we observed 17 users install WebCloud on different browsers and operating systems. These users connected a total of 585 times to the proxy, and browsed 2,069 photos with an average session time of 18 minutes. 539 (or 26%) of these photos were served from another WebCloud client. While this fraction of WebCloud is lower than in our simulations, it is likely due to our deployment environment: For the simulation, we considered what would happen if Facebook deployed WebCloud to an entire region; in our real-world deployment, we had to manually recruit people, so our social network coverage is substantially lower. However, the deployment demonstrates that WebCloud is feasible with today's OSN sites and web browsers.

V. Related work

A. Content distribution networks

CDNs like Akamai, Limelight, Adero and Clearway offload work from the original website by delivering some or all of the content to end users, often using a large number of geographically distributed servers located in different ISPs. While most CDNs are operated in a centralized manner, systems such as Coral [37], [38] have been built which use decentralized architectures to accomplish the same task. These solutions are well-used, but they generally rely on resources donated by governments and universities, and are therefore not self-sustaining in the long run.

Other approaches have been explored allowing end users to participate in CDNs, including Akamai's NetSession [9], a client-side application that assists in content distribution to other Akamai clients, and FireCoral [8], a browser plug-in that participates in the Coral network. While the goals of both of these systems are similar to WebCloud, both require the user to download and install a separate application or plug-in, limiting their applicability and userbase.

Other work [15] has proposed to reduce long latency and high loss on paths between users and the Facebook servers by serving content locally. Unlike our work, they do not utilize end users' capability but instead deploy local TCP proxies and cached content on those proxies.

Additionally, recent work [39] has demonstrated that information flow patterns over social networks (called *social cascades*) can be leveraged to improve CDN caching policies. This work is complementary to ours, and further demonstrates the importance of leveraging properties of the social network in future CDN designs. Finally, other recent work [40] has examined the benefits of allowing ISPs to assist CDNs in making content delivery decisions. This approach is similar in spirit to ours, but focuses on optimizing the server selection strategies employed by ISPs today.

B. Peer-to-peer systems

WebCloud can be viewed as approximating peer-to-peer (p2p) content exchange though web browsers. Much previous work has focused on building standalone p2p systems for content storage and exchange [41]–[43] or avoiding the impact of flash crowds [44]. In those systems, clients are generally have little choice of the peers where their data is stored, raising security, privacy, and reliability issues. To address these concerns, researchers have examined mechanisms for storing content between friends. For example, CrashPlan and Friendstore [45] provide cooperative backup systems that allow users to store their data on trusted peer nodes. However, unlike WebCloud, almost all p2p systems assume a full networking stack, preventing a browser-based deployment.

VI. Conclusion

In this paper, we took a step towards decentralized web-based content exchange by introducing WebCloud. WebCloud makes novel use of established web technologies to allow clients to help serve content to other clients, keeping the content exchange local and providing savings for both the site OSN and the ISP. The result is that WebCloud is able to serve as a drop-in component on sites like Facebook, Flickr, and MySpace. Microbenchmarks, simulations, and

[9] Our real-world deployment was covered under Northeastern Institutional Review Board application #10-07-23.

a small-scale deployment on Facebook demonstrated that WebCloud works well in practice, and that WebCloud holds the potential to serve over 40% of the content requests, even when used in conjunction with a CDN.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers, Bimal Viswanath, Max Marcon, and Ansley Post for their helpful comments. We also thank the users of our WebCloud prototype for their assistance in evaluating WebCloud. This research was supported by NSF grant CNS-1054233 and an Amazon Web Services in Education Grant.

REFERENCES

[1] "Facebook and YouTube dominate workplace traffic and bandwidth," http://www.scmagazineuk.com/facebook-and-youtube-dominate-workplace-traffic-and-bandwidth/article/168082/.
[2] "Alexa Top 500 Global Sites," http://www.alexa.com/topsites.
[3] G. Cormode and B. Krishnamurthy, "Key Differences between Web 1.0 and Web 2.0," *First Monday*, vol. 13, no. 6, 2008.
[4] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Watch Global, Cache Local: YouTube Network Traffic at a Campus Network - Measurements and Implications," in *MMCN*, 2008.
[5] N. Kennedy, "Facebook's photo storage rewrite," http://www.niallkennedy.com/blog/2009/04/facebook-haystack.html.
[6] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "YouTube Traffic Characterization: A View from the Edge," in *IMC*, 2007.
[7] P. Vajgel, "Needle in a haystack: efficient storage of billions of photos," http://www.facebook.com/note.php?note_id=76191543919.
[8] J. Terrace, H. Laidlaw, H. E. Liu, S. Stern, and M. J. Freedman, "Bringing P2P to the Web: Security and Privacy in the Firecoral Network," in *IPTPS*, 2009.
[9] "Akamai NetSession," http://www.akamai.com/client.
[10] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta, "PeerSoN: P2P Social Networking – Early Experiences and Insights," in *SNS*, 2009.
[11] "Diaspora*," http://www.joindiaspora.com/.
[12] "Most Popular Extensions :: Add-ons for Firefox," https://addons.mozilla.org/en-US/firefox/extensions/?sort=users.
[13] "Adblock Plus : Statistics for Adblock Plus," https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/statistics/usage/?last=30.
[14] "Mozilla Metrics Report, Q1 2010," https://wiki.mozilla.org/images/e/ed/Analyst_report_Q1_2010.pdf.
[15] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao, "Exploiting Locality of Interest in Online Social Networks," in *CoNEXT*, 2010.
[16] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *INFOCOM*, 1999.
[17] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel, "You are who you know: Inferring user profiles in Online Social Networks," in *WSDM*, 2010.
[18] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *IMC*, 2007.
[19] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao, "Understanding Latent Interactions in Online Social Networks," in *IMC*, 2010.
[20] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *EuroSys*, 2009.
[21] J. Leskovec, L. Backstrom, and J. Kleinberg, "Meme-tracking and the dynamics of the news cyle," in *KDD*, 2009.
[22] "Facebook Statistics," http://www.facebook.com/press/info.php?statistics.
[23] "4,000,000,000 $\ll$ Flickr Blog," http://blog.flickr.net/en/2009/10/12/4000000000/.
[24] X. Cheng, C. Dale, and J. Liu, "Statistics and Social Network of YouTube Videos," in *IWQoS*, 2008.
[25] A. Clauset, C. R. Shalizi, and M. E. Newman, "Power-law distributions in empirical data," *SIAM Review*, vol. 51, no. 4, 2009.
[26] M. Arlitt and T. Jin, "Workload Characterization of the 1998 World Cup Web Site," *IEEE Network*, vol. 14, no. 3, 2000.
[27] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: Provider Portal for Applications," in *SIGCOMM*, 2008.
[28] "Akamai," http://www.akamai.com/.
[29] "W3C WebStorage," http://www.w3.org/TR/webstorage/.
[30] T. Stein, E. Chen, and K. Mangla, "Facebook Immune System," in *EuroSys*, 2011.
[31] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," in *IEEE S&P*, 1998.
[32] M. J. Freedman and R. Morris, "Tarzan: A Peer-to-Peer Anonymizing Network Layer," in *CCS*, 2002.
[33] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Growth of the Flickr Social Network," in *WOSN*, 2008.
[34] "Hitwise Intelligence - Robin Goad - UK," http://weblogs.hitwise.com/robin-goad/2010/08/facebook_accounts_for_1_in_6_uk_page_views_has_it_reached_saturation_point.html.
[35] A. N. Joinson, "Looking at, looking up or keeping up with people?: Motives and use of Facebook," in *CHI*, 2008.
[36] M. Cha, A. Mislove, B. Adams, and K. P. Gummadi, "Characterizing Social Cascades in Flickr," in *WOSN*, 2008.
[37] M. J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing content publication with Coral," in *NSDI*, 2004.
[38] M. J. Freedman, "Experiences with CoralCDN: A Five-Year Operational View," in *NSDI*, 2010.
[39] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft, "Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades," in *WWW*, 2011.
[40] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann, "Improving Content Delivery Using Provider-aided Distance Information," in *IMC*, 2010.
[41] L. P. Cox and B. D. Noble, "Samsara: honor among thieves in peer-to-peer storage," in *SOSP*, 2003.
[42] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A cooperative internet backup scheme," in *USENIX ATC*, 2003.
[43] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *SOSP*, 2001.
[44] A. Stavrou, D. Rubenstein, and S. Sahu, "A Lightweight, Robust P2P System to Handle Flash Crowds," in *ICNP*, 2002.
[45] D. N. Tran, F. Chiang, and J. Li, "Friendstore: Cooperative online backup using trusted nodes," in *SNS*, 2008.