Demo: Siskin: Leveraging the Browser to Share Web Content in Disconnected Environments

Samuel Sudar
University of Washington
Seattle, WA
sudars@cs.uw.edu

Matt Welsh
Google, Inc.
Seattle, WA
mdw@mdw.la

Richard Anderson
University of Washington
Seattle, WA
anderson@cs.uw.edu

ABSTRACT

Schools in the developing world frequently do not have high bandwidth or reliable connections, limiting their access to web content. As a result, schools are increasingly turning to Offline Educational Resources (OERs), employing purpose-built local hardware to serve content. These approaches can be expensive and difficult to maintain in resource-constrained settings. We present Siskin, an alternative approach that leverages the ubiquity of the web browser to provide a distributed content access cache between user devices on the local network. We demonstrate that this system allows access to web pages offline by identifying the browser as a ubiquitous platform.

1 INTRODUCTION

The web has tremendous potential to enable education for users in emerging markets. With an increasing amount of free or open-use educational content becoming available online (Khan Academy, Wikipedia for Schools, etc.), schools in developing regions can bring vast quantities of human knowledge directly into the classroom. However, the web is built with the assumption of fast, free, and always-on connectivity. This presents a substantial challenge to schools in developing regions, which often have slow or intermittent Internet connectivity, such as a flaky dialup connection [4]. While modern web standards make it possible to develop sites that can be used offline (e.g., using ServiceWorkers to persist content in the browser), this approach does not scale to the vast quantity of legacy web content. The conventional approach to working around this problem involves caching static snapshots of web content, typically on dedicated hardware [1, 3]. However, this approach poses substantial logistical and cost challenges for schools in emerging markets. A typical edge cache box costs upwards of several hundred dollars, which can be prohibitive, and still requires manual maintenance and updates to software and content [9]. Schoolteachers, especially in developing countries, are not system administrators and cannot be expected to maintain esoteric hardware and software to support classroom web use.

Our key observation is that modern web browsers have the capability to support everything needed to enable automatic, distributed caching of web content that can be shared across multiple users on a LAN. While the basic idea is not new—distributed caches have been explored since the foundations of the web [8, 11, 12]—our work leverages three key insights that represent a practical, deployable solution for real-world users today. First, conventional HTTP caching is inadequate for supporting true offline access to Web content. Given that most web pages (and many resources that they depend on) are uncacheable [2], and the degree to which websites use dynamic content fetched, e.g., using AJAX, standard HTTP caches cannot guarantee that a given page will be fully cache-resident and hence usable by a user without an Internet connection. Second, separating the functionality and maintenance of the cache from the device and software used day-to-day implies that functionality will erode quickly. The caching solution should be integrated directly into the device and software (e.g., the browser) that the user interacts with, to ensure it does not introduce a single point of failure and does not suffer from neglect. And third, modern protocols (such as Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD)) enable automatic discovery and sharing of resources across a LAN segment, leveraged in products such as the Chromecast media-streaming device and others. This can enable seamless sharing across multiple users without overhead for configuration and management.

In this paper we describe *Siskin*, an approach to distributed snapshot caching of web content fully integrated into the Chrome browser. To use Siskin, the user simply needs to install a Chrome App and Extension. Siskin allows users to locally save snapshots of web content that they browse, and it automatically shares those snapshots with other users on the LAN. While browsing, Siskin automatically discovers snapshots of pages hosted by other users on the network, and it retrieves those snapshots over the local network. Siskin is built using existing Chrome App and Extension APIs to perform web content snapshotting, storage, network discovery, and peer-to-peer content transfer.

2 SYSTEM DESIGN

Siskin provides a seamless distributed cache of web page snapshots available to any device on the same LAN segment; this will typically be a single classroom or a goup of classrooms. Every peer in the Siskin network hosts web page snapshots, which can be discovered and fetched by other peers on the network, avoiding the need for fetching content from a slow or intermittent Internet connection. Siskin achieves this using a combination of local page snapshotting and caching; peer discovery using mDNS; peer-to-peer snapshot fetching using Web Real-Time Communication (WebRTC); and a Chrome App to provide the UI.

CHANTS'17, October 20, 2017, Snowbird, UT, USA.
 ISBN 978-1-4503-5144-7/17/10.
DOI: http://dx.doi.org/10.1145/3124087.3124088

2.1 Snapshotting Content

The fundamental unit in Siskin is the rendered page. This creates a one to one mapping between a top-level URL and a resource resident in the cache. This approach effectively snapshots the rendered web page that results from a visit to a URL. Unfortunately there is not yet a standard for distributing web pages that perfectly recreates a connected experience. The most widespread support is for MHTML, which takes a snapshot of the DOM, inlining external resources like images. This has the benefit of being a single hermetic file, making distribution simple, and of being well-supported by browsers. When a saved page is viewed, it is fetched from the peer, saved to disk, and displayed in a browser tab. MHTML snapshots are saved manually by users. When visiting a page, clicking the Siskin Extension icon in the Chrome toolbar saves the page as MHTML and adds it to the cache using the Chrome App and Extension APIs. For security reasons, we elected to keep snapshotting a manual process.

Although widely supported, a notable shortcoming of MHTML is the fact that it does not support JavaScript execution. For this reason responsive sites and web apps are not well-suited to MHTML. An alternative approach might be to ignore cache-control headers and simply cache all resources, as in [5] and [10]. We find the single file, hermetic nature of MHTML to be a preferable distribution mechanism. Resource-level caches are best suited to configurations where a cache sits between a machine and the origin server, allowing individual requests to be handled in flight. Saving pages as an MHTML resource simplifies distribution by allowing alternative configurations, including look-aside caching behavior where a user is informed before navigation that a local load will succeed. Employing MHTML allows content to be added to the cache as logical units and distributed with associated provenance. It is explicit that MHTML is only a snapshot, not a stale page served from a cache.

Siskin does not try to support web app behavior that requires complex interactions with a server. Email applications, for example, are not handled. Siskin aims only at operations where a page can be displayed without needing to interact with a server after the time the page is saved.

2.2 Peer Discovery

A discovery component is necessary to find peers running Siskin on the local network. We accomplish this by employing mDNS and DNS-SD. These zero-configuration protocols are a standard solution to the problem of network service discovery; they are employed by many services, including Chromecast media streaming devices [6, 7]. mDNS uses multicast UDP to issue DNS queries to the local network, while DNS-SD specifies how to use DNS records as a hierarchical database for service discovery. Using both together, clients can discover peers running a service and resolve an IP address and port combination to connect to the service. Chrome Apps provide an API to issue and respond to UDP requests, making an implementation of mDNS and DNS-SD straight forward.

2.3 Content Discovery

Local content discovery takes one of two forms. In the first, a list of peers on the network is presented via the App UI. Upon selecting a peer, the list of pages saved by that peer is presented to the user much like listing the contents of a directory. In the second, regular browsing behavior is augmented to inform users of locally available content. Locally available URLs are styled with icons to indicate their availability. This process is referred to as "cache coalescence". To accomplish this, peers disseminate Bloom filters, as in [8], representing the URLs they have cached locally. The App component of Siskin maintains this information, allowing instances to locally determine if a given link is available from a peer. Links on a page are annotated to show that they are available.

Querying the App component for coalescence information does not inhibit browsing. To measure query latency, we constructed four HTML pages with 1, 10, 100, and 1,000 outbound links. We also generated a synthetic cache directory containing 10 peers, each of which contained 1,000 pages. The cache directory was already present in memory, as if peer state had already been communicated. The query latency is the time to look up all of the outbound links on the page in the cache directory. The mean results over 100 runs are shown in Figure 1.

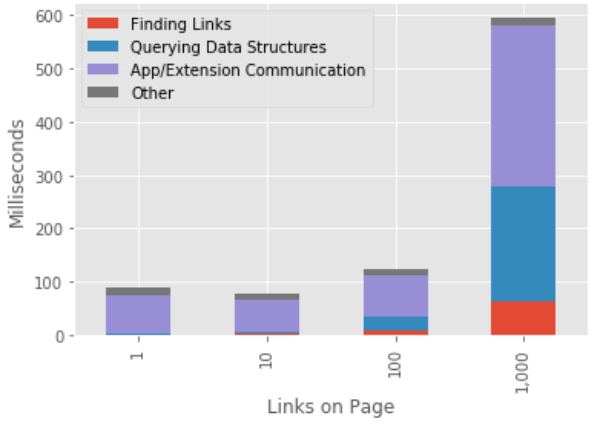

Figure 1: Mean times to query for URLs available on the network of 30 peers, each with 1,000 pages.

Our prototype of Siskin uses a fairly simplistic cache coalescence strategy—each peer exchanges a Bloom filter of locally-cached URLs with every other peer on a periodic basis (every 60 seconds). We perform this via unicast, where each peer sends a Bloom filter to every other peer. More efficient schemes, for example, leveraging multicast, are also possible.

We analyze a theoretical network of between 2 and 50 peers, each of which is caching 1,000 pages locally. Each peer encodes the list of cached URLs into a Bloom filter with a target false positive rate of 0.001, which requires 1,798 bytes. We consider an 802.11b network, as might be expected at a rural school, with an aggregate TCP throughput of 5.9 mbps. Fully distributing cache state requires each peer to communicate their state to every other peer, resulting in 4.4 MB that must move across the network. This would require 6.0 seconds if all 50 peers joined the network at the same time. Figure 2 shows the bandwidth requirements of our unicast strategy for different refresh rates. This assumes that an initial distribution has been completed and the Siskin peers are periodically informing peers of their content by completely redistributing their Bloom filters.

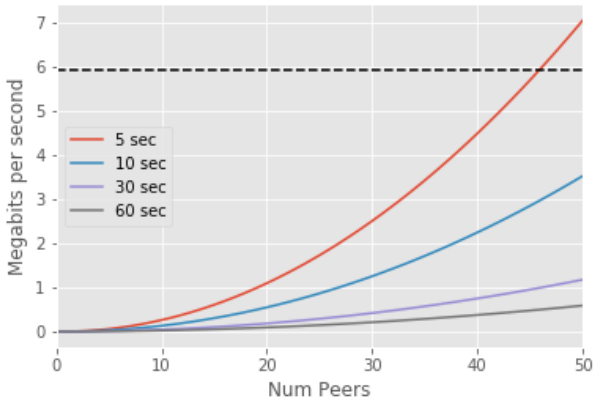

Figure 2: Bandwidth consumed by fully redistributing cache directory information via a unicast mechanism. Bandwidth is estimated at different refresh rates as additional peers join the network, each hosting 1,000 pages. The dashed line shows a theoretical maximum throughput of 5.9 mbps.

2.4 Security and Privacy

We identify three main areas relevant to security and privacy in Siskin: 1) confidentiality of shared content; 2) secure communication between peers; and 3) integrity of cached content.

The first area relates to the confidentiality of content. Our approach of snapshotting pages makes this problem non-trivial. If a user elects to snapshot their email client or social media wall, for instance, they run the risk of leaking potentially private information to others on the network. We elected to make snapshotting pages require a manual action from the user. In our prototype, content is guaranteed not to be shared until hitting the extension icon and electing to save content, allowing users explicit control over what content is available. Additional defenses could include blacklisting social media sites and implementing access control, allowing snapshots to be private or restricted to specific users.

The second area relates to secure communication. Siskin should support both encrypted and authenticated communication. The WebRTC transport mechanism ensures that communication is encrypted, but in our prototype we do not authenticate this connection. However, it is possible to perform authentication via a protocol akin to Secure Simple Pairing (SSP), which is used to pair Bluetooth devices. It provides a way to both exchange an encryption key and authenticate that the exchange was not subject to a man in the middle attack. SSP includes a numeric comparison mode by which two users compare several numeric digits, checking for equality. This exploits physical proximity of two devices and is appropriate for Siskin, where users are expected to be on the same LAN.

The final area, integrity of content, is not provided by our prototype. Snapshotting and sharing are performed by untrusted peers, meaning that integrity guarantees stemming from the use of HTTPS are lost. Snapshots could be tampered with, or sophisticated peers could falsely claim that a fabricated snapshot originated from a specific domain. Integrity guarantees could be maintained by adding a level of trust to peers and cryptographically signing snapshots. An alternative could be to use a third party service to generate and sign snapshots. These approaches would require additional infrastructure and are not things we currently support.

3 RELATED WORK

A more conventional practice on challenged networks is to install an HTTP cache. Efforts at increasing the efficiency of caching solutions have included making multiple caches cooperate [8, 11]. The C-LINK system performs cooperative caching by using a coordinating proxy to store resources using clients' local storage [10]. If caching conventions are ignored and stale resources are served, as in [10] and [5], resource-level HTTP caching remains inadequate as it assumes that devices on a network are at least partially connected, even if over a challenged backhaul. By design a cache first checks locally, and in the event of a miss it goes to the network. Since caching occurs at the network level, there is no way to inform the user of a miss. If a request misses the cache, the user might like to know that the request to the origin server will fail or will take more time than they are willing to wait. Conventional cache models do not allow this.

Our work is most similar to [5], who implement aggressive HTTP caching as a Firefox extension. That project does not coordinate between machines, preventing users from benefitting from peer caches on the network. It is also aimed at accelerating browsing behavior, e.g. through aggressively prefetching links on a page from the internet, rather than on distributing content. For these reasons their system is not well-suited as a platform for OERs.

4 CONCLUSION

Siskin demonstrates that OERs at schools in the developing world do not require additional hardware or purpose-built devices. Standalone solutions can be replaced or complemented with existing, ubiquitous infrastructure—the browser—to give disconnected users access to the more than four billion pages that exist on the web.

REFERENCES

[1] 2017. Critical Links C3. http://c3.critical-links.com/. (2017).
[2] 2017. HTTP Archive. http://httparchive.org/. (2017).
[3] 2017. Intel CAP. https://www-ssl.intel.com/content/www/us/en/education/products/content-access-point.html. (2017).
[4] Fundación Sergio Paiz Andrade. 2016. Evaluation Report: Assessing the use of technology and Khan Academy to improve educational outcomes in Sacatepéquez, Guatemala. (2016).
[5] Jay Chen, David Hutchful, William Thies, and Lakshminarayanan Subramanian. 2011. Analyzing and Accelerating Web Access in a School in Peri-urban India. In *WWW '11*. ACM, New York, NY, USA, 443–452.
[6] Stuart Cheshire and Marc Krochmal. 2013. *DNS-based service discovery*. Technical Report.
[7] Stuart Cheshire and Marc Krochmal. 2013. *Multicast dns*. Technical Report.
[8] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. 2000. Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol. *IEEE/ACM Trans. Netw.* 8, 3 (June 2000), 281–293.
[9] Nicci C. L. Gafinowitz. 2016. Digital Library Appropriation in the Context of SubSaharan Countries: The Case of eGranary Digital Library Implementation. In *ACM DEV '16*. ACM, New York, NY, USA, Article 29, 4 pages.
[10] Sibren Isaacman and Margaret Martonosi. 2009. The C-LINK system for collaborative web usage: A real-world deployment in rural Nicaragua. In *Workshop on Networked Systems for Developing Regions*.
[11] Radhika Malpani, Jay Lorch, and David Berger. 1995. Making World Wide Web caching servers cooperate, In Proceedings of the Fourth International World Wide Web Conference. (December 1995), 107–117.
[12] Alec Wolman, M. Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin, and Henry M. Levy. 1999. On the Scale and Performance of Cooperative Web Proxy Caching. In *SOSP '99*. ACM, New York, NY, USA, 16–31.