Prizm: A Wireless Access Point for Proxy-Based Web Lifelogging

Jimmy Lin, Zhucheng Tu, Michael Rose, and Patrick White
David R. Cheriton School of Computer Science
University of Waterloo, Ontario, Canada
{jimmylin,michael.tu,msrose,ps2white}@uwaterloo.ca

ABSTRACT

We present Prizm, a prototype lifelogging device that comprehensively records a user's web activity. Prizm is a wireless access point deployed on a Raspberry Pi that is designed to be a substitute for the user's normal wireless access point. Prizm proxies all HTTP(S) requests from devices connected to it and records all activity it observes. Although this particular design is not entirely novel, there are a few features that are unique to our approach, most notably the physical deployment as a wireless access point. Such a package allows capture of activity from multiple devices, integration with web archiving for preservation, and support for offline operation. This paper describes the design of Prizm, the current status of our project, and future plans.

1. INTRODUCTION

At the intersection of the lifelogging [5] and quantified self [13] movements, it is now fairly mainstream for individuals to monitor and record their daily activities in a consistent and structured manner for subsequent analysis. For example, wearable devices count how many steps we take each day and monitor the quality of our sleep, a multitude of apps are available to track the food we consume and other activities, and various sensors measure aspects of our environment. A noticeable gap is the lack of principled mechanisms to record, monitor, and analyze our online activities. Questions an individual might want to answer include:

- How much time have I spent on Facebook and other social media sites in the last week?
- Do I tweet more often on weekends than I do on weekdays? Do I need to self censor?
- How many Google queries have I issued in the last month? How many of them were repeated?
- Am I relying too much on Wikipedia for research?
- Do I browse the web more from my tablet than from my laptop? And when?
- How many YouTube videos have I watched in the last month? How much time have I spent watching political satire?
- Are there any unexpected connections made by services I use that I should be aware of?

Many internet services already know the answer to these questions, at least for specific verticals (e.g., Google, Facebook, etc.). Somewhat ironically, these services know more about users' behaviours than most users themselves. Nevertheless, there does not exist a device that comprehensively captures a user's web activities—until now.

In this paper, we present Prizm, a prototype lifelogging device that comprehensively captures a user's web activity. Prizm is a wireless access point deployed on a Raspberry Pi, an inexpensive computer the size of a deck of playing cards. The device is designed to be a substitute for the user's normal wireless access point. An integrated software stack on the Prizm device proxies all HTTP(S) requests from devices connected to it and Prizm records all activity it observes. Although this particular design is not entirely novel (see discussion in related work), there are a few features that are unique to our approach, most notably the physical deployment as a wireless access point—which addresses multi-device usage, integration with web archiving, and support for offline operation. This paper describes the design of Prizm, the current status of our project, and future plans.

2. BACKGROUND AND RELATED WORK

Our work is most closely related to that of d'Aquin et al. [3], who described a tool for capturing web activity via an HTTP proxy and presented an analysis of one individual's activities over a span of 2.5 months. We adopt a similar technical approach based on proxying, which offers a much more detailed record of user activities, as compared to, for example, solutions based on browser plugins (e.g., [17]).

Although there exist even more comprehensive forms of activity capture, for example, techniques that record interactions with all applications on a computer [2], we believe that our focus on web activity has one major advantage. Web activity is more homogeneous (compared to arbitrary application usage), thus facilitating more insightful analyses across multiple devices (see below). Given the amount of computer usage that occurs within a web browser and the proliferation of web services (Gmail, Facebook, etc.) that

LTA'16, October 16 2016, Amsterdam, Netherlands

ISBN 978-1-4503-4517-0/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2983576.2983581

have become an ingrained part of our daily lives, we believe that restricting our focus to web activity does not create substantial gaps in coverage, even if our approach does not yield exhaustive capture of *all* online activity.

Beyond the proxy-based activity capture of d'Aquin et al. [3], we claim three unique features:

Multi-device integration. Users access the web today from a multitude of devices: desktops, laptops, tablets, as well as smartphones. This means that any view of a user's web activity restricted to a single device will be incomplete. Most devices, however, connect to the web via Wi-Fi within a fixed geographic area (e.g., home, office, etc.). Therefore, implementing activity monitoring in a wireless access point can capture a multi-device view of a user's web activity. Indeed, such a deployment package also makes it possible to monitor small groups of individuals that share the same wireless access point (e.g., family members, roommates, etc.).

Web archiving. Web archiving refers to the systematic collection and preservation of web content for future generations. The web has become an integral part of our daily lives and captures our "collective memory", making it an important part of our cultural heritage. Since web pages are ephemeral and disappear with great regularity [12], the only sure way of preserving web content for posterity is to proactively crawl and store portions of the web.

A longstanding question in web archiving has been: which sites should we crawl and how frequently? Web archivists have proposed a few different answers. The Internet Archive has been collecting and storing web content since 1996, via broad but relatively shallow crawls. A loosely-organized network of national, academic, and other libraries build special collections that focus on particular domains or subjects—these are generally deeper, more focused, and take place with greater frequency. A third approach is to drive collection efforts based on social media, for example, archive pages that are posted in tweets. Milligan et al. [11] provided a recent comparison and discussion of these strategies.

To this discussion, we contribute a fourth answer: let us archive content that people actually use. Since our lifelogging device is already monitoring user activities, it is easy to also *store* a copy of the content that passes through the proxy. Cheap storage makes this practical today. Lin [8] showed the feasibility of storing and accessing web archives on a Raspberry Pi, but did not study the actual content capture mechanism. This work, in effect, fills that gap via proxy-based, usage-driven web archiving.

There are, of course, substantial privacy concerns for such a web archiving mechanism, but we can imagine a few sensible solutions. We could adopt an opt-in panel system whereby users are compensated for their contributions, with full disclosure and transparency on what exactly is captured. This would be no different from mechanisms in place today where companies monitor users' general web browsing habits (e.g., comScore) or television viewing preferences (e.g., Nielsen ratings) for compensation. Another plausible scenario is a framework whereby users voluntarily contribute (i.e., upload) their personal archives to a trusted third party—for example, a cultural heritage institution—which then aggregates the captured content (suitably anonymizing and deduplicating data in the process). Safeguards can be taken in this aggregation process to protect the identities of individual users.

Offline web usage. The integration of web activity monitoring and personal web archiving presents an opportunity to support offline web usage. As an example, previous studies have shown that a significant fraction of users' search behaviour on the web consists of "refinding" [16], or searching for pages they had encountered before—in which case, why not reduce latency by returning the contents of the previously-cached versions? Indeed, researchers have already explored running full-text search engines on mobile phones [1] in support of this scenario. Related, Lin [9] demonstrated that it is possible to provide a full-text searchable copy of Wikipedia on a Raspberry Pi, serving its contents to anyone nearby via either Wi-Fi or Bluetooth. This means that many queries can be answered without going out to the live web.

One might argue that the ubiquity of internet connections today makes such scenarios uncompelling. This might be true in the developed world, but in many developing countries, internet access is unreliable and often subjected to usage caps [10], which might make offline web usage more attractive. For example, consider rural villages in India [15] or Easter Island, where only satellite internet is available. However, even in the developed world, mobile internet remains subjected to data limits in one form or another (e.g., monthly data quotas). This is particularly bothersome if a user travels frequently and wishes to stream videos away from a Wi-Fi access point that offers, for the most part, unlimited bandwidth.

Let us consider the following scenario: the user signals to Prizm what YouTube videos she is interested in watching, which triggers the device to download and store the videos locally. This uses exactly the same mechanism that is used for web archiving, except the scenario is better described as proactive caching. When the user leaves for her trip, she simply takes the Prizm device with her physically. The device is small enough to store in a backpack, and can be equipped with a battery to provide reasonable operating life. While traveling (for example, on a train), she can now watch the cached YouTube videos without consuming valuable mobile bandwidth. Since Prizm remains a wireless access point (albeit without a corresponding uplink to the internet), it can transparently serve content it has cached. We might think of this as a new take on the decades-old idea of "file hoarding" [6, 14] to create the illusion of web access without requiring an internet connection. Given that users' online behaviour exhibits some degree of locality (e.g., reoccurring search queries or frequent references to Wikipedia), we believe that a reasonable offline web experience is possible with only modest amounts of storage.

3. PRIZM ARCHITECTURE

Prizm is a prototype lifelogging device for recording and analyzing web activity deployed as a wireless access point on a Raspberry Pi, an inexpensive computer about the size of a deck of playing cards that has become popular in the educational technology and maker communities. Prizm refers to both the device and the software running on it, but references should be clear from context.

Our current implementation runs on two generations of Raspberry Pi hardware: The Raspberry Pi 2 (Model B), released in early 2015, features a 900 MHz quad-core ARM Cortex-A7 CPU with 1GB RAM. The Raspberry Pi 3 (Model B), released in early 2016, features a 1.2GHz 64-bit quad-

Figure 1: Prizm on a Raspberry Pi 2 (Model B). (Image Credit: Wikipedia)

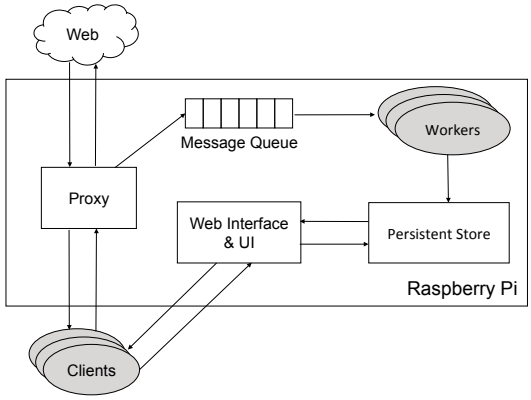

Figure 2: Architecture overview of Prizm. Components within the rectangular box reside on the Raspberry Pi.

core ARMv8 CPU, also with 1GB RAM. For both, internal storage is provided by a microSD card, and the device has four USB ports for connecting peripherals. An external display can connect via an HDMI port, but once properly configured, Prizm functionalities can be accessed remotely. A Raspberry Pi 2 is shown in Figure 1.

In this section, we describe the Prizm software infrastructure that supports multi-device integration, web archiving, and offline web usage. There are five distinct components in our design (see Figure 2), including the proxy for channeling HTTP requests, the workers for processing metadata from the HTTP requests and responses, the message queue for handling communication between the proxy and workers, the persistent store for capturing web activity, and an HTTP server that provides a web portal allowing users to interact with captured data on the device.

3.1 Wireless Access Point and Proxy

Before any traffic goes through the proxy, the Raspberry Pi itself needs to be capable of acting as a wireless access point. For this to work, the Raspberry Pi needs to have a WLAN interface for servicing the wireless access point (`wlan0`, backed by a Wi-Fi USB dongle on the Raspberry Pi 2 or built-in 802.11n WLAN on the Raspberry Pi 3) and another interface that actually connects to the internet (`eth1`, backed by the Raspberry Pi's built-in Ethernet port). We assign a range of private IP addresses to the wireless network (`192.168.42.x`) and configure the `wlan0` interface to use IP addresses in the wireless network range.

As a wireless access point, the Raspberry Pi must be capable of assigning IP addresses to clients that wish to connect to it. This is accomplished with a DHCP server using the `wlan0` interface. We use `hostapd` as the access point daemon, which also handles authentication. Traffic can be routed between `wlan0` and `eth1` using `iptables`. We add a rule to the Network Address Translation (NAT) table with the MASQUERADE action to replace the sender IP address of packets originating from the clients with the IP address of the `eth1` interface.

For the actual proxy software we use mitmproxy,[1] a man-in-the-middle proxy for HTTP with support for SSL written in Python. This system has well-documented, easy-to-use APIs, and based on initial experiments, performance is reasonable. As an example, mitmproxy provides a simple response callback API for content adaptation, where HTML is modified on the fly. One of the uses of this feature is to inject content from results cached locally for offline usage.

Although HTTP traffic can be intercepted and modified out of the box using mitmproxy, doing the same with HTTPS traffic requires additional steps. HTTPS adds a SSL/TLS encryption layer on top of HTTP, which is designed to prevent interception by a third party between the client and the server. When an SSL connection is established, the client needs to verify the certificate provided by the server to establish its identity. The client does this by checking if it trusts the certificate or if the certificate is trusted by one of the Certificate Authorities (CAs) that it trusts. Note that any CA can essentially issue certificates for any website. By installing the mitmproxy CA certificate on the client, the client will in general trust any dummy certificates for the SSL sites that the client visits. Hence, Prizm can be used to monitor HTTPS traffic in the same way it is used to monitor regular HTTP traffic.

Recently, Public Key Pinning Extension for HTTP (certificate pinning) was proposed in RFC 7469 [4] to combat CAs mis-issuing certificates. In certificate pinning, websites tell the client to remember ("pin") a set of cryptographic identities for a period of time, and during this time the client can only accept certificate chains that include at least one key whose fingerprint matches that of a pinned identity. This means that mitmproxy's certificates will not be accepted by applications that employ certificate pinning. Currently, the only way to use such applications is to tell the mitmproxy to ignore these domains so they are not intercepted by the proxy.

To integrate mitmproxy with the wireless access point, we redirect traffic on ports 80 and 443 for HTTP and HTTPS traffic, respectively, to the proxy using `iptables`. One current weakness is that mitmproxy does not support websockets, but we are currently developing workarounds.

[1] https://mitmproxy.org

3.2 Message Queue

As HTTP and HTTPS requests and responses are channeled through the proxy, we implemented a callback via an API provided by mitmproxy to insert corresponding messages into a message queue for further processing. Each request/response pair generates one message: these messages encapsulate metadata about the requests and responses, including the time, domain name/IP address, URL path, referer, user-agent, content type, and content length.

We adopted an architecture based on a message queue because it allows us to decouple producers (the proxy) from downstream consumers; this is sometimes called "event sourcing"[2] and is a popular architecture today for building distributed systems [7]. Such an architecture allows potentially I/O-intensive operations such as writing to a persistent store (more details below) to proceed asynchronously, without affecting the performance of the proxy. This architecture also allows us to add an arbitrary number of workers to handle multiple tasks in a loosely-coordinated manner.

We currently use RabbitMQ[3] as our message queue, since it has a large community, rich features, and can run reasonably well on a Raspberry Pi. We use the Python client `pika`[4] to communicate with RabbitMQ.

3.3 Workers and the Persistent Store

In the current implementation of Prizm, we have one type of worker that retrieves messages from the queue, performs extraction and normalization of the data, and persists the output to a MySQL database running on the Raspberry Pi. An example of normalization is entity resolution of the domain name: `www.google.com` and `google.com` should sensibly be treated as the same entity. In Section 3.5, we describe additional workers for supporting web archiving and offline operations.

We use MySQL to store detailed metadata about each request/response pair that passes through the mitmproxy, hereafter referred to as just a request. Using the request time and the domain and URL path of the request, we can answer a variety of questions about the distribution (across services and time) of web usage. With simple regular expressions and additional post-processing, we can extract query strings for web searches. From the referer, we can construct a graph of how users navigate from one page to another during browsing sessions. The user-agent can be used to determine which device connected to the access point sent the request. Finally, the content type and content length can be used to determine what types of media users request and how much data they consume. Various analyses can be formulated in terms of SQL queries, which we hide behind visualizations (more details later).

The database also stores category labels for popular websites, such as "News" for `cnn.com` and "Social Media" for `twitter.com`. Users can define custom categories so that they can understand their web activities based on aggregate groups rather than individual websites. Questions such as "how long did I spend on social media websites compared to educational websites" can be answered easily based on these labels. Users can edit the domain categorization via the web application discussed in the next section.

[2]http://martinfowler.com/eaaDev/EventSourcing.html
[3]https://www.rabbitmq.com
[4]https://github.com/pika/pika

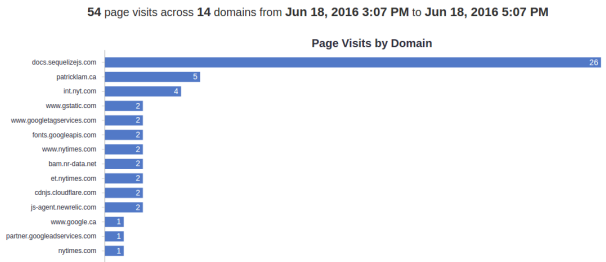

Figure 3: A simple visualization of data collected by Prizm. Domains visited during the browsing session are represented by bars, and the length of each bar shows the number of page visits to that domain.

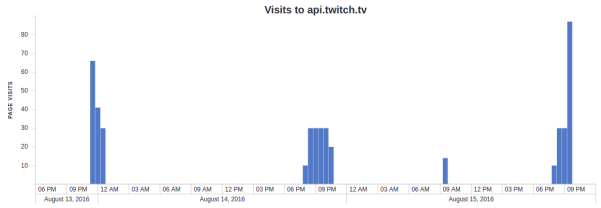

Figure 4: Visualization of page visits for one domain. The x-axis denotes time, while the y-axis shows the frequency of visits. This particular graph aggregates page visit frequencies at 30-minute intervals.

3.4 Web Portal

The final component of Prizm is the *portal*—a web application served directly from the Raspberry Pi that allows users to interact with and configure the system.

Upon first connecting to a Raspberry Pi running Prizm, the user visits the portal in her browser to conduct the necessary setup. The setup consists of downloading and trusting the SSL certificate generated by mitmproxy, and setting a custom SSID and password for the wireless access point.

Once the user has connected to Prizm and starts using the access point, the primary function of the portal is to provide visualizations and analyses of web usage. Users can select a date range and then view different graphs describing web activity within that date range. Figure 3 shows an example of a visualization of the number of page visits to all domains from a short browsing session. Underlying data is retrieved from the persistent store via a REST API written using Flask,[5] a lightweight Python library for building web applications. The UI is built with the React JavaScript framework.[6] This design decouples the back-end and front-end in order to accommodate data sources generated by other Prizm workers (see Section 3.5).

We provide a few additional examples of visualizations that we are developing. Figure 4 displays the number of page visits to a particular domain over time. A Prizm user can harness such a graph to see what time of day she usually visits a particular site; she can also observe the background activity of specific applications over the course of time.

[5]http://flask.pocoo.org
[6]https://facebook.github.io/react/

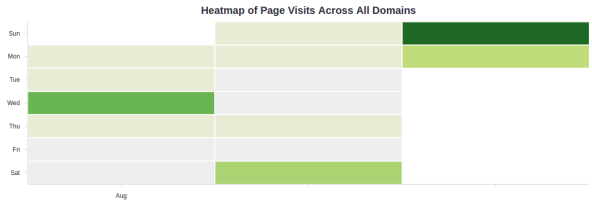

Figure 5: An example heatmap visualization of overall web usage. Darker colours indicate periods of heavier usage (i.e., more total page visits).

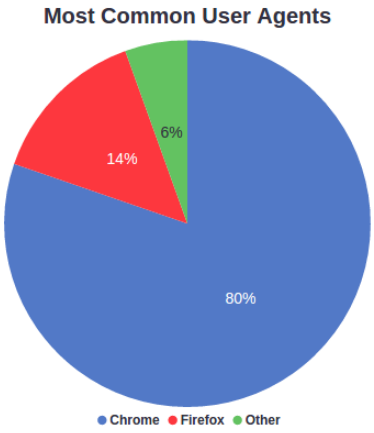

Figure 6: An example pie chart depicting which agents a user most commonly employs to make page requests.

Figure 5 shows an example of a heatmap, similar to the one used to track contributions on GitHub.[7] This heatmap depicts the total number of page visits made on a given day, where darker colours indicate periods of higher activity. A Prizm user can quickly discern, for example, which day of the week has the highest web usage (and when).

Figure 6 breaks down the various agents used to access the web, and allows a user to see exactly *how* she accesses online content. As we previously discussed, multi-device integration is a major benefit of deploying Prizm on a Raspberry Pi (see Section 2)—by drilling down into different agent types (not shown here), a user is able to determine, for example, how much web browsing she does from her smartphone versus her laptop.

Beyond analytics and visualizations, user preferences can be configured from the portal. A user can exclude domains from the analyses, manage customized categorizations for the domains to provide higher-level summaries, and define customized visualizations of web activity.

As part of future work, when web archiving functionalities are integrated into Prizm, the portal will serve as the entry point to the offline search engine, providing users with an interface to manage and locate content stored on the Raspberry Pi (see Section 3.5). In our architecture, the portal front-end is not directly coupled to any back-end data source, making it easy to extend.

[7]https://github.com/blog/1360-introducing-contributions

3.5 Web Archiving and Offline Usage

The current implementation of Prizm has only one type of worker, to persist records of web activity into MySQL. However, as we discussed in Section 2, the Prizm concept provides support for web archiving and offline web usage, which we discuss in this section. These capabilities can be implemented via other types of workers that consume messages in the message queue, and our architecture supports loosely-coupled integration of additional functionalities with minimal coordination. In particular, our architecture allows flexible worker scheduling and deferred execution of certain tasks for performance considerations. As a simple example, we can delay I/O-intensive operations to a time when the device is mostly idle.

We have investigated web archiving and offline operations in an early experimental prototype of the system, although these functionalities have yet to be integrated into the current Prizm implementation. We discuss in more detail:

Web archive creation. We are currently exploring two alternative designs for web archiving. In the first, given a URL (or a set of URLs) that we wish to archive (retrieved from the message queue), we can invoke the utility `wget` to capture the content, along with all stylesheet, image, and related assets. Data can be written in WARC format, which is a web archiving standard—thus supporting interoperability with a large ecosystem of web archiving tools. In the alternative design, we could insert all content payload (not just metadata) directly into the message queue, so that the worker can directly persist the content without refetching the pages over the network. Data in this design can similarly be stored in WARC format. In both cases, it may not be desirable to archive everything that the user browses, e.g., videos or certain sensitive pages such as banking information, so we must implement some type of whitelist or blacklist mechanism. This is possible in both designs.

In the first design we are essentially redownloading URLs, even though the contents have already passed through the mitmproxy. The second design introduces a potential performance bottleneck in the message queue. In both designs, since web archiving is considered a secondary functionality in Prizm, it makes sense to perform the operation when the system is mostly idle, as to not affect the core user experience. This is likely easier with the first design, as there is a danger of overflowing the queue in the second design and the larger message sizes may interfere with the latency of other operations. The redownloading approach is more conducive to performing web archiving in batch and allows us to generate efficient compressed output while avoiding needless file manipulations, e.g., aggregating small files into larger bundles (in the alternative design there will always be pressure to clear the message queue relatively quickly to prevent overflow). Redownloading, however, means that the archived pages may differ from the actual page that the user viewed. Overall, it is unclear which design is better; we are exploring both implementations and will evaluate the alternatives empirically.

Offline browsing of cached content. Captured web content (stored as WARC files) can be "replayed" using a system such as `pywb`,[8] which allows a user to browse the archive pages and navigate links to contemporaneous pages (to the extent that they are also captured in the web archive). This

[8]https://pypi.python.org/pypi/pywb/

functionality has already been demonstrated on a Raspberry Pi [8], and therefore we anticipate straightforward integration into Prizm. When the device is not connected to the internet, the mitmproxy provides an API that lets us intercept HTTP requests and provide responses from cached content in `pywb`, thus supporting transparent offline usage.

Archive indexing and search. The creation of the web archive can trigger full-text indexing of page content using the Lucene search engine. Search capabilities can be provided in two modes: first, when the user explicitly wishes to search previously-encountered content, and second, transparently redirecting web searches to local data when Prizm is disconnected from the internet.

4. USAGE EXPERIENCE

We have completed the initial Prizm prototype as described in Section 3. A subset of the authors have begun to use the device in their daily activities—we are in what is colloquially known as the "dogfooding" phase. After the prototype matures, the next stage of our project is a user study in which we actually deploy the device to end users.

Overall, the user experience for most websites we visit and applications we use connected to Prizm is not markedly different from the experience when connected to a home wireless network directly, in terms of response load times and preserving site features. We performed a simple page load timing experiment to record the response load times quantitatively. To simulate the load times for web pages typical users will actually visit, we use the homepages of the top 100 domains in the ALEXA ranking.[9] Our experiments compare the response times of accessing these homepages in the following configurations:

- using a home wireless router,
- using either the Raspberry Pi 2 (Model B) or Raspberry Pi 3 (Model B) as a wireless router only, and
- using the same devices with the entire Prizm software stack in its current implementation.

For each configuration, three trials were performed. Figure 7 shows our results in terms of average load times, measured in seconds; the bars are annotated with relative load time increases compared to the baseline of using the home wireless router. These figures show that there is indeed latency overhead that is introduced by Prizm, which is the cost of the additional features provided by the device (specifically, 41% increase in load times on the latest Raspberry Pi 3). Reducing this overhead is the focus of ongoing work, although we note that much of this problem will likely be solved with newer generation hardware over time—compare the performance difference between the Raspberry Pi 2 and Raspberry Pi 3. However, the measurable overhead of Prizm seems to be tolerable based on our own usage experience.

In terms of website features, we are able to perform bandwidth-intensive activities such as watching YouTube videos or streaming Spotify music seamlessly. However, due to the lack of support for websockets, web services that depend on websockets cannot be used smoothly through the proxy. A workaround we've currently implemented is to exclude a list of user-configurable domains from the proxy so users can still

[9]http://www.alexa.com

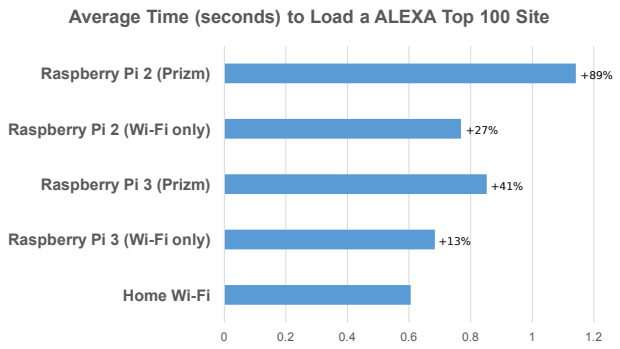

Figure 7: Average time to load an ALEXA top 100 site in seconds, comparing different configurations with home Wi-Fi. Bars are annotated with percent increase over home Wi-Fi. Lower is better.

enjoy these services without being affected. We are working on a more robust solution that can record activities for all domains. Another limitation of Prizm we have encountered concerns the security issues we discussed in Section 3.1 regarding the trust of certificates generated by the mitmproxy CA. Handling the untrusted certificates is also a part of our future work in hopes of delivering an even more seamless user experience.

5. CONCLUSIONS

In today's world, big internet companies such as Google, Facebook, and Netflix know more about our online presence than we do ourselves. We aim to change that: Prizm is an inexpensive lifelogging device that can comprehensively record our web activities. Week, months, or even years later, we can recollect, reminisce, and reflect on our previous activities [5]. The current Prizm prototype has an extensible architecture to support additional capabilities in the future, including web archiving, offline usage, and full-text indexing. We are enthusiastic about the future potential of this device in particular, and more generally, the concept of monitoring one's own online activities as a form of lifelogging.

6. REFERENCES

[1] A. Balasubramanian, N. Balasubramanian, S. J. Huston, D. Metzler, and D. J. Wetherall. FindAll: A local search engine for mobile phones. *CoNEXT*, 2012.

[2] G. Barata, H. Nicolau, and D. Gonçalves. AppInsight: What have I been doing? *AVI*, 2012.

[3] M. d'Aquin, S. Elahi, and E. Motta. Personal monitoring of web information exchange: Towards web lifelogging. *WebSci*, 2010.

[4] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP. RFC 7469, Internet Engineering Task Force, 2015.

[5] C. Gurrin, A. F. Smeaton, and A. R. Doherty. LifeLogging: Personal big data. *Foundation and Trends in Information Retrieval*, 8(1):1–107, 2014.

[6] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, 1992.

[7] M. Kleppmann. *Making Sense of Stream Processing*. O'Reilly, Sebastopol, California, 2016.

[8] J. Lin. Scaling down distributed infrastructure on wimpy machines for personal web archiving. *WWW Companion*, 2015.

[9] J. Lin. The sum of all human knowledge in your pocket: Full-text searchable Wikipedia on a Raspberry Pi. *JCDL*, 2015.

[10] A. Mathur, B. Schlotfeldt, and M. Chetty. A mixed-methods study of mobile users' data usage practices in South Africa. *UbiComp*, 2015.

[11] I. Milligan, N. Ruest, and J. Lin. Content selection and curation for web archiving: The gatekeepers vs. the masses. *JCDL*, 2016.

[12] A. Ntoulas, J. Cho, and C. Olston. What's new on the web? the evolution of the web from a search engine perspective. *WWW*, 2004.

[13] M. Swan. The quantified self: Fundamental disruption in big data science and biological discovery. *Big Data*, 1(2):85–99, 2013.

[14] C. Tait, H. Lei, S. Acharya, and H. Chang. Intelligent file hoarding for mobile computers. *MobiCom*, 1995.

[15] W. Thies, J. Prevost, T. Mahtab, G. T. Cuevas, S. Shakhshir, A. Artola, B. D. Vo, Y. Litvak, S. Chan, S. Henderson, M. Halsey, L. Levison, and S. Amarasinghe. Searching the world wide web in low-connectivity communities. *WWW*, 2002.

[16] S. K. Tyler and J. Teevan. Large scale query log analysis of re-finding. *WSDM*, 2010.

[17] D. Zagorodnov, L. Brenna, C. Gurrin, and D. Johansen. WAIFR: Web-browsing attention recorder based on a state-transition model. *CAMA*, 2006.