On Browser-Level Event Logging

Owen Bell,[†] Mark Allman, [*] and Benjamin Kuperman[§]

TR-12-001

January 2012

Abstract

In this paper we offer an initial sketch of a new vantage point we are developing to study "the Web" and users' interactions with it: we have instrumented the Web browser itself. The Google Chrome browser provides an API to developers that allows the building of extensions to the base functionality. As part of this system, Chrome allows developers to add listeners to various *browser events*. Our extension adds listeners that log these events. We discuss the data we obtain from Chrome, our method for addressing privacy issues in the collected data, and initial findings from observing a small set of real users' Web browsing activities. The findings are modest in absolute terms, but serve to show the efficacy of our monitoring approach.

† Case Western Reserve University, Cleveland, OH 44106
* ICSI, Berkeley, CA 94704
§ Oberlin College, Oberlin, OH 44074

This work was partially supported by funding provided to ICSI through National Science Foundation grant CNS: 0722035 ("Architectural Support for Network Trouble-Shooting"). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors or originators and do not necessarily reflect the views of the National Science Foundation.

1. INTRODUCTION

The "web" is built on top of HTTP [8], a flexible protocol that allows information to be gathered from myriad places (servers, databases, content-delivery networks, advertising systems, etc.) and integrated into a logical whole for users. Our understanding of the web largely comes from (*i*) anecdotes, (*ii*) network-level passive observation of users' browsing behavior, and (*iii*) rudimentary probing of web servers. Arguably, each of these has offered much insight into the system over the years. However, there are still gaps in our understanding due to our incomplete vantage points. For instance, passive observation allows a wealth of information to be harvested about actual user-produced web transactions, but can derive little insight about encrypted transactions. Further, determining which transactions are combined to form the logical whole that the user views is often tricky when stitching the story together from packet traces. Even tracking a "user" is difficult since often users get network addresses dynamically and/or share an address simultaneously using network address translators (NATs). On the other hand, synthetic probing allows us to piece together an understanding of how the overall system is structured (how CDNs are utilized, which protocol features are implemented, etc.), but offers no insight into how these features and structures manifest when faced with actual user activity.

In this paper we offer an initial sketch of a new vantage point we are developing to study "the web": we have instrumented the web browser itself. The Google Chrome browser provides an API to developers that allows for building various extensions to the base browser functionality. As part of this system, Chrome allows developers to add listeners to various *browser events*. We simply add listeners that log the time the event happens and the arguments passed to the listener. This model of event recording—with no analysis in the browser—closely follows a model that has proven useful in other areas of network data collection (e.g., RouteViews [12], intrusion detection [14]).

As described in § 2, Chrome's set of events run the gamut from creating windows and tabs to bookmarking a web page. By recording these events and then analyzing the resulting logs we can create an accurate user-centric picture of web browsing. We believe this viewpoint can shed significant light on various aspects of user behavior that are not apparent from network traces. For instance, from passive observation we cannot determine anything useful about encrypted HTTP transactions, whereas from our browser logs we can gain insight into this traffic (see § 3.1).

In addition to simply characterizing user behavior in web browsers, we believe that a fine-grained log of a user's actions may well provide operational benefits in terms of troubleshooting. That is, when a web site is not loading for some reason (local network problem, CDN outage, DNS issue, web server under attack, etc.) the user's behavior may well be different than when all components are working as expected. If we can detect this from the browser event logs, we can then design browsers to also detect such events and perhaps alert operators. While our data analysis shows that developing these causal event chains is possible, in this preliminary work we do not tackle this troubleshooting issue in detail.

This paper is a modest initial foray into the space. Our Chrome extension is in its formative stages, our data collection is small in scope (15 users), and our data analysis is preliminary. We hope to use this short paper to show the promise of the technique and engage with the community on further interesting questions we might ask and further data we might collect.

2. LOGGING FRAMEWORK

Chrome provides an interface to allow users to create *extensions*, which are small programs that modify the behavior of the browser. We created an extension that listens for relevant browser events and simply records the event, a timestamp, and all ancillary information provided to the listener. The event handling and logging will add overhead to the operation of the browser. However, since our extension only saves information on each event it is hard to envision it causing user-perceptible overhead, and in fact subjectively we do not notice it nor did any of the users who installed the extension and reported back to us. In addition, we do the *bare minimum* in the browser, therefore reducing the overhead will necessarily reduce the amount of data collected.

2.1 System Architecture

Our logging extension registers a set of event listeners with Chrome. When the extension receives an event, it saves single line of text representing the event. Each record includes: (*i*) a timestamp, (*ii*) the event name (e.g., "`tabs.onCreated`"), (*iii*) the Chrome-assigned window and tab identifiers for the tab where the event occus, (*iv*) the URL and (*v*) any additional information based on the given event type.

Chrome makes extensive use of sandboxing to protect the host OS from potentially malicious software making it difficult or impossible to get direct access to the file system to save logs. However, the HTML 5 standard includes a specification for a persistent data storage of key-value pairs in web browsers called `localStorage` [10] which is where we stored our log. At present, users emailed the generated log manually. While this worked fine as a proof-of-concept, near-term future work will entail automatic submission and log rollover capabilities.

Finally, note that not every browser action generates

an event—e.g., there is no event associated with browser startup. This can complicate some of our analyses as we will now have to infer what happened rather than directly observing it. That said, the application-layer log provides a better basis for such inference than network-level or host-level logs.

At present the events logged by our Chrome extension can be broken into four broad classes, as follows:

- **Windows:** creation, removal, focus on/off
- **Tabs:** creation, removal, update (e.g., starting or stopping a web page load), attach/detatch to a window, focus on/off
- **History:** visiting a web page component, removing URL from browser history
- **Bookmarks:** creating, changing, removing, importing from another browser/file, being manually rearranged, being sorted by the browser

We further log various events caused by our extension itself—e.g., when the user clears the log, when logging is stopped/started, that the user's secret (see § 2.2) changed, etc.

2.2 Privacy Issues

When building a log of web browsing activity for use by someone besides the user, privacy concerns abound. We leveraged three principles to address these concerns in developing our extension: (*i*) users control their own logging, (*ii*) users can readily observe what is being logged, and (*iii*) logs can be optionally encoded to obscure the most sensitive portions (e.g., URLs visited). Our goal is to protect the privacy of users such that they will be comfortable turning over their data to us for analysis while also retaining enough rich logging that we can mine the data for useful results. Our small pilot tests have shown reasonable success in this goal, however broader tests may of course cause an evolution in our techniques. We next discuss our implementation of the privacy principles listed above.

Logging Control: Within our extension users can start and stop logging with a single button press. While these logging on/off events are recorded such that we are able to understand that the logs are incomplete, nothing else is recorded about any facet of browsing while the user has explicitly stopped logging. Further, Chrome disables extensions while the user is in "incognito" mode and therefore we log nothing about users' activities during such periods.

Understanding Logged Information: Next, we provide users with an understanding of what information has been logged and therefore what will be returned to the researchers. All logging is done in plain text such that the user can directly see what will be returned. Additionally, event names are specified and parameters labeled. While not all users may wish to poke through the log in detail, we believe providing the option to view what the researchers are requesting provides users with confidence that we are not collecting any sort of personal data in a clandestine fashion. Similarly, the code is readily available for inspection—which we do not expect many users to do, but does present an open posture that we believe is useful.

Obfuscate Sensitive Data: Finally, even with the above considerations some of the information we wish to log is sensitive. For instance, the events expose URLs accessed, page titles, etc. Therefore, we give users the option to obfuscate their logs. The extension picks a random secret S when it is first executed.[1] The secret is stored within the browser but never exposed outside the local host (i.e., in the log). We then define an obfuscating function as:

$$G(x) = MD5(x + S) \tag{1}$$

We then log page titles as $G(title)$. URLs are encoded to retain their structure. Consider the URL:

http://foo.com/script.cgi?args

Rather than running $G()$ across the entire URL we encode the components as:

$$\text{http://}G(foo).\text{com/}G(script.cgi)?G(args)$$

This mechanism follows the spirit of prefix-preserving IP address anonymization [16], whereby the goal is to obfuscate the values themselves, but retain the structure of the values. By encoding in this fashion we protect privacy as each user's log is different (due to S). But, we retain certain aspects of the URL, such as the application protocol, the top-level domain, etc. As always, obfuscating is a tradeoff between privacy and research usefulness [13]. We believe we have struck an appropriate balance, and the users in our initial data collection effort seemed satisfied with our methods (some asked specific questions about the high-level mechanism we sketched during recruiting and ultimately were satisfied in our safe-guards).

We also stress that we are not trying to anonymize the data in such a way that it could be released publicly (and therefore should be able to withstand all manner of attacks). Rather, we are obfuscating the data such that the users are comfortable turning over the data to a small group of well-known researchers. We expect that this will lower the obfuscation burden for most users.

3. DATA ANALYSIS

We next turn to an initial analysis of the browser-event logs we collected from the 15 users who used our extension. Our goal in this paper is to illustrate the sorts of analyses that we can undertake with the data provided by our framework and show some results

[1]The user can enter their own secret, if desired.

User	Events (K)	Dur. (days)
$\mathcal{U}_0$	10.5	5.4
$\mathcal{U}_1$	6.0	4.6
$\mathcal{U}_2$	12.5	2.8
$\mathcal{U}_3$	11.5	8.1
$\mathcal{U}_4$	10.0	15.1
$\mathcal{U}_5$	12.1	6.6
$\mathcal{U}_6$	3.6	7.5
$\mathcal{U}_7$	7.0	5.8

User	Events (K)	Dur. (days)
$\mathcal{U}_8$	8.2	6.4
$\mathcal{U}_9$	3.9	6.9
$\mathcal{U}_{10}$	5.1	6.1
$\mathcal{U}_{11}$	2.6	6.1
$\mathcal{U}_{12}$	1.2	6.1
$\mathcal{U}_{13}$	1.9	2.0
$\mathcal{U}_{14}$	2.7	13.8
Total	98.8	103.3

Table 1: User characteristics.

that are suggestive that this measurement methodology holds promise. In particular, we note that neither our sample of users nor the length of our observations are enough to draw general conclusions. Further, one can readily think of many additional analyses that could be conducted, however, we present only a sample to show the power of the framework and leave a more comprehensive analysis as future work.

To help us both ensure our Chrome extension was working and to provide data for initial analysis we recruited colleagues and friends we knew to be running Chrome.[2] The participants enabled logging for 2.8–15.1 days and logged between 1.2K–12.5K events (see Table 1). In total the logging encompasses over 100 days worth of user activity and captures nearly 100K events. The table shows the users have different browsing rates. For instance, $\mathcal{U}_2$'s log spans less than half the time captured in $\mathcal{U}_{12}$'s log and yet contains an order of magnitude more events. Finally, we note that our data has an obvious "geek bias" given who was asked to install and run our extension. As discussed above, we do not find this problematic for our initial foray because we are vetting the method more than drawing general conclusions. However, this must be addressed in future collection efforts.

3.1 Hidden Performance Issues

As sketched above, one of the benefits of logging inside an application is gaining visibility into events that are obfuscated on-the-wire. HTTPS is widely used to encrypt web sessions to thwart attackers who can snoop a network—e.g., on an open wireless network—from stealing sensitive information and/or engaging in man-in-the-middle attacks. While this use of encryption is obviously beneficial, the drawback is that researchers and operators lose understanding of what is happening in the network. We are not advocating clear-text transmission for research benefits, but note that the loss of visibility does have a cost in terms of our understanding of the network and therefore seek to regain enough visibility to keep our understanding current. In particular, we find that across our user sample HTTPS comprises 1%–70% of the web pages loaded, with most users falling within the 10%–35% range.

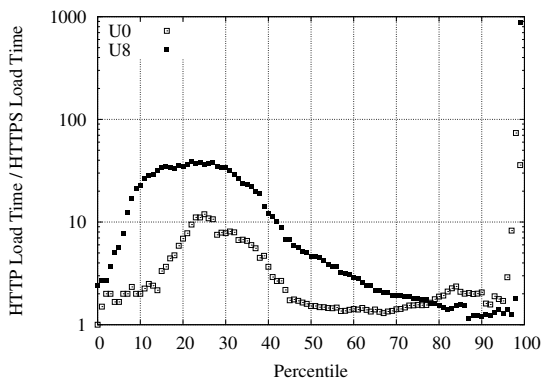

Figure 1: Relative performance of HTTP and HTTPS across percentiles.

Chrome fires events when web page loading is initiated and completed. This allows us to assess the load time without manually correlating all the disparate objects that form a page as an on-the-wire analysis would entail. We find that load times for secure web pages[3] are generally *shorter* than for normal HTTP-loaded pages (using means, medians, comparing quartiles, etc.). While we find this result counter-intuitive, it indeed holds across all users in our dataset. As an illustration, we calculated the ratio between the duration of HTTP and HTTPS page loads for each percentile of each user's distribution of load times. Figure 1 shows the results for two users. All ratios are greater than one, indicating HTTP loads are slower than HTTPS. The differences are more pronounced for $\mathcal{U}_8$ than for $\mathcal{U}_0$. We have not yet uncovered the reason why HTTPS pages load more rapidly. The reasons could range from content—e.g., a difference in average page sizes[4]—to structural—e.g., secure sites frequented by users being geographically close such as a Blackboard-like system on campus. Our future work will entail logging much more information to dig into these kinds of puzzles, but already one can see our approach bearing fruit in terms of previously unattainable results.

[2]Note, due to the varying abilities of web browsers some users in our sample did note that they used other browsers for certain activities (e.g., sites written specifically for Internet Explorer). Therefore, in some cases we did not capture *all* a given users web browsing and in fact our sample might be biased. We leave an assessment of this bias as future work, but note that our initial data collection effort has shown us the importance of providing users with a survey to better understand the context of the returned data.

[3]Note: just because the main web page uses HTTPS does not mean all components are delivered via HTTPS.

[4]While it is natural to assume correlating with the page size would be trivial, the browser does not provide size information to event listeners. One piece of future work will be using additional means to access this information (e.g., via querying the cache).

3.2 Tracking Users Across Time and Space

Next, we note that monitoring within browsers can be used to more accurately track *an individual user's* activity across both time and network connectivity. While a network-level monitor may get a reasonable—but not complete—view of a user's web activity, with today's device mobility a given network vantage point will nearly assuredly only view slices of the user's overall activity. Roughly half the users in our sample employed Chrome on a laptop that regularly changed connection point during data collection (with the remainder being run from a fixed device). Using out-of-band information (logs of an often- and automatically-accessed service) we were able to determine that the second author spent his working hours connected to two different networks during his collection period for 23% and 77% of the time, respectively. This illustrates that network vantage points at either location would miss a non-negligible portion of his traffic.[5] A network's viewpoint is further complicated by dynamic addresses and NATs which can easily cause users to be aggregated together without an easy way to untangle them.

Note that digging activity from network-level data is cumbersome and time consuming and can therefore be a logistical impediment to gathering a large amount of data. However, logging this same activity within a browser is a simple matter of recording state that already exists. Therefore, monitoring on an end host or in an application itself can provide a *comprehensive* and *low cost* view of a user's activities. While this has been previously observed [9] we are able to get an even more detailed perspective given our integration with the application. A final point is that using a particular network vantage point (a la a recent study of HTTP we conducted [4]) includes natural bias in terms of the user population. While our modest user sample does not materially change this effect the approach of collecting data from web browsers offers the chance to sample users much more broadly than can be done from any one vantage point.

As an illustrative example, we note that our sample of users loaded roughly 17K URLs during the monitoring period. Of these, 13K were loaded only once. Figure 2 shows the mean and median number of URL visits per day for URLs visited more than once as a function of user (sorted by mean number of URL visits). Due to mobility and encrypted traffic, this figure could be at best only approximated without browser level monitoring. The figure shows that of the pages loaded more than once, the median number of retrievals is fewer than once per day across most users. However, the mean can be multiples of the median indicating that the distribution is skewed and that some pages are loaded a large number of times. Finally, we note there are many additional avenues for tracking users' patterns across time with infrastructure.

[5]This also suggests that we may need to add an active probe to our extension—with the agreement of each user—to aid in understanding when users change their point of attachment.

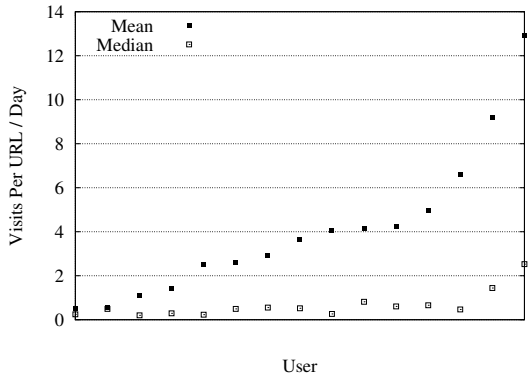

Figure 2: Visits per URL per day.

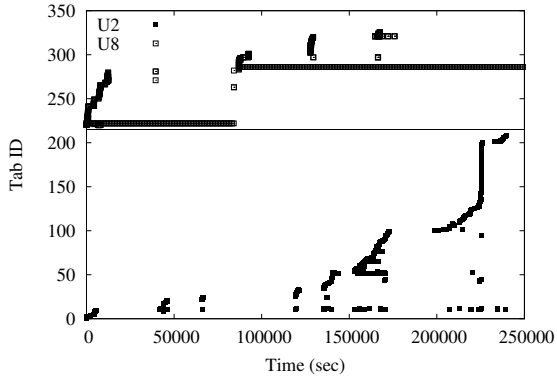

Figure 3: Timeline of completed downloads.

3.3 Modeling and Troubleshooting

As developed above, one of the issues with attempting to model or troubleshoot web traffic via network observation is that the vantage point lacks enough context to fully understand the chain of causal events that triggered a particular web object to be fetched. At best the observed behavior is an estimate. Figure 3 provides a view of web browsing developed from our browser logs for two sample users. In the figure, each point represents a completed web page download with the x-axis representing the time and the y-axis representing the tab in which the web page was loaded. We plot $\mathcal{U}_2$'s full history and the beginning of $\mathcal{U}_8$'s history (for presentation purposes).[6] Using a network vantage point the dimension illustrated on the y-axis would be unavailable. Further, with dynamic address assignment and NATs a

[6]Note: The tab IDs used on the y-axis are arbitrary and we offset $\mathcal{U}_8$'s ID such that they are greater than 220 while $\mathcal{U}_2$'s tab IDs are all less than 220.

host's exhibited network behavior may be aggregated with other hosts and hence obfuscated. This figure shows some of the sorts of causal aspects a browser-based log can illuminate. For instance, we see that $\mathcal{U}_8$ has a long-running tab open that fairly continuously retrieves data—but, this seems invariant of whether the user is actively browsing the web. Analysis of the logs indicates this is an automatic updating of a particular web page every 15 minutes. However, such automated retrieval should not be confused with the user's interactive browsing.

Additionally, we can see that both users open many short-lived tabs in close succession. We find that in 10% of the cases a tab is opened within 2.2 seconds of the last tab open event. In a number of cases we see a progression of tabs being used as time passes, indicating that users are opening links in new tabs as they browse. With the browser log we can understand which page triggered each new tab by noting the focus when tabs are created. Therefore we can build a chain of events that can then be used to model browsing in a way that traffic analysis cannot—yielding a richer model that better follows actual user behavior. Further, such a behavioral history can be the basis of detecting anomalous behavior—e.g., uncharacteristic automatic page fetching.

As a concrete example, consider troubleshooting network issues by assessing whether a web page has completely loaded or not. A network-based monitor can attempt to track the main page and the set of included objects to determine if they all load successfully. However, if they do not load properly the network-based monitor is left with only an indication that there *may* be a problem. In our dataset, however, we found 294 instances where the user started a new retrieval before the previous retrieval in the same tab finished. In these cases an external monitor may arrive at the conclusion that a problem could be occurring, but the browser itself has enough information to understand that there is no problem.

3.4 User-Interface Issues

While our focus is on observing network-related browsing behavior our approach lends itself to studying the web browser's user interface, as well. Browser extensions have been previously employed to record so-called "click streams" the represent user's interactions with the application itself (e.g., see [7] as a recent example). Such data can be used to understand how people use various browser features and also to point out common usage scenarios that may lead to better interfaces.

As a modest example, Figure 4 shows two aspects of how users employ tabs. For each user we determined the length of time a tab was open for all tabs that the user both opened and closed during our logging.[7] We then calculated various percentiles of these per-user distributions and then plot the CDF of those values across users in the left-hand plot. This plot shows a range of tab usage. Many tabs are open at most several minutes. However, we also observed a non-negligible number of tabs that were open for longer than one day. Further, we note that user behavior in terms of tab lifetime is clearly quite broad—spanning three orders of magnitude in time at each percentile shown on the figure. A second aspect of tab usage we are able to assess is the number of web pages loaded within a given tab. I.e., do people use a tab for each web page they load? Or, are tabs used repeatedly for various tasks? The right-hand portion of Figure 4 shows the distributions of number of web pages loaded per tab for three sample users. $\mathcal{U}_0$ and $\mathcal{U}_9$ roughly represent the upper and lower bound behavior across all users. We include $\mathcal{U}_{11}$ as a data point in-between. The plot shows that some users such as $\mathcal{U}_9$ load very few web pages per tab—i.e., 90% of the tabs being used to load one or two web pages. Others such as $\mathcal{U}_0$ re-use tabs a bit more—i.e., the 90^{th} percentile being ten web pages per tab.

While interface issues are not our focus, collecting data in this area is a nice bonus to bring more data to these issues. For instance, the aforementioned browser study in [7] encompasses click-streams from a mere 21 users which illustrates the need for more data and therefore opportunistic collection and synergistic partnerships will be part of our future work.

4. RELATED WORK

The literature is filled with work exposing the operation of "the web" from various angles. We believe our work is complementary to the existing body of work and offers another viewpoint on the process. Below we discuss the relationship of our work to various categories of previous work (and provide but a small sample of the vast references in each category).

Much of the previous work characterizing the web was conducted using passive traffic monitoring (e.g., [1–3, 5]). Such studies provide the foundation on which our understanding of web traffic is built and the methodology is useful in a number of ways. For instance, a large user population can be studied via a single vantage point. However, such monitoring has several issues including (*i*) vantage point bias, (*ii*) lack of insight into encrypted traffic (10% of web connections viewed at one institutional border in a recent study [4]), (*iii*) sampling bias due to device mobility and (*iv*) traffic ambiguity caused by NATs and dynamic address assignment. Each of these leaves either ambiguity or an interesting subset of the traffic under-studied.

[7] I.e., we discounted tabs open when the logging started or stopped.

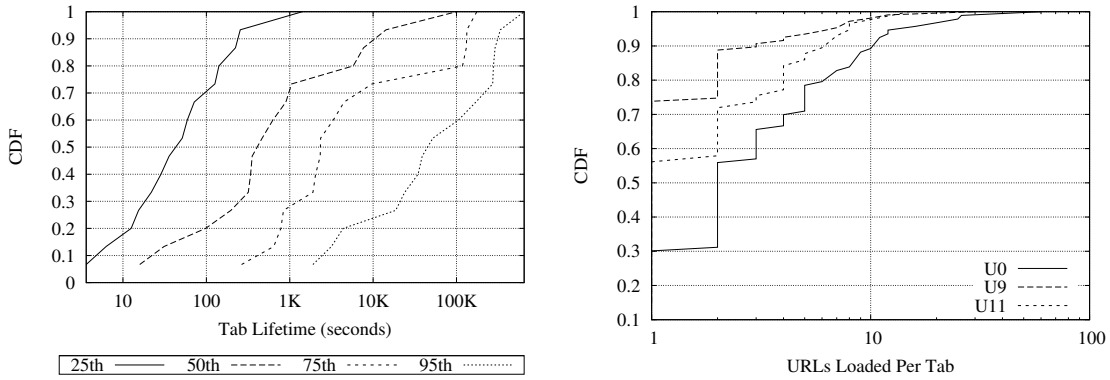

Figure 4: Tab characteristics: tab lifetime (left), pages loaded in tab (right).

Another closely related set of previous work involves monitoring on the end host and/or the application itself. Researchers have previously proposed making applications monitor certain aspects of their behavior and/or performance for their own benefit. For instance, many BitTorrent clients assess performance from various peers and use this as input in selecting where to next request pieces of a particular download (e.g., see the BitTyrant work [15]). In addition, application agnostic end-host monitoring has previously been conducted (e.g., [9, 11]). Our approach of logging application-specific details in web browsers can compliment these more general application agnostic measurements.

Finally, we note that end-host measurements have been proposed for aiding intrusion detection by exposing events that cannot be observed by network-based sensors due to encryption or lack of context (e.g., see [6]). Our logs could form the basis of a mechanism for feeding into such systems.

5. CONCLUSIONS AND FUTURE WORK

We make two main contributions in this paper. First, we develop a Chrome extension for logging information about users' web activity. Our logging strategy is designed around simple logging of browser events without analysis. In addition, our strategy is built on several principles that take into account user privacy which is crucial to any effort to collect data from users. Our second contribution is a number of initial analyses of a small set of data (from 15 users) that show the promise of our monitoring apparatus to observe web browsing details that have been previously unavailable to the community's monitoring efforts. This paper represents only a small initial step and our future work entails harvesting additional information from the browsers and expanding our user sample in terms of number and breadth.

Acknowledgments

Initial development on our browser plugin was conducted by Siyang Wu. Discussions with and data from a number of people aided the work, including: Kenneth Atchinson, Ethan Blanton, Tom Dooner, Hua Jiang, Aaron Kanter, Tu Ouyang, Jesse Rowsell, Brain Stack, Sipat Triukose and Kevin Woods. The work was supported in part by NSF grant CNS-0722035. Our thanks to all!

References

[1] M. Arlitt and C. Williamson. Web Server Workload Characterization: The Search for Invariants (Extended Version). *IEEE/ACM Transactions on Networking*, 5(5), Oct. 1997.

[2] P. Barford and M. Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS*, pages 151–160, July 1998.

[3] P. Barford and M. Crovella. Measuring Web Performance in the Wide Area. *Performance Evaluation Review: Special Issue on Network Traffic Measurement and Workload Characterization*, Aug. 1999.

[4] T. Callahan, M. Allman, and V. Paxson. A Longitudinal View of HTTP Traffic. In *Passive and Active Measurement Conference*, Apr. 2010.

[5] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, Dec. 1997.

[6] H. Dreger, C. Kreibich, R. Sommer, and V. Paxson. Enhancing the Accuracy of Network-based Intrusion Detection with Host-based Context. In *Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, 2005.

[7] P. Dubroy and R. Balakrishnan. A study of tabbed browsing among mozilla firefox users. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, CHI '10, pages 673–682, New York, NY, USA, 2010. ACM.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. RFC 2616.

[9] F. Giroire, J. Chandrashekar, G. Iannaccone, K. Papagiannaki, E. Schooler, and N. Taft. The Cubicle Vs. The Coffee Shop: Behavioral Modes in Enterprise End-Users. In *Passive and Active Measurement Conference*, Apr. 2008.

[10] I. Hickson. Web storage. Last call WD, W3C, Dec. 2009.

[11] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. HostView: Annotating End-host Performance Measurements with User Feedback. In *ACM Sigmetrics HotMetrics Workshop*, June 2010.

[12] U. of Oregon RouteViews Project. http://www.routeviews.org.

[13] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. In *ACM SIGCOMM Computer Communication Review*, volume 36, Jan. 2006.

[14] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proc. of the 7th USENIX Security Symposium*, Jan. 1998.

[15] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent? In 4^{th} *USENIX Symposium on Networked Systems Design & Implementation*, 2007.

[16] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, 2002.