2008

AjaxTracker: A Tool for High Fidelity Characterization of Ajax Applications

Myungjin Lee

Sumeet Singh

Ramana Rao Kompella
Purdue University, kompella@cs.purdue.edu

Report Number:
08-019

AjaxTracker: A Tool for High Fidelity Characterization of Ajax Applications

Myungjin Lee
Sumeet Singh
Ramana Rao Kompella

CSD TR #08-019
July 2008

AjaxTracker: A tool for high-fidelity characterization of Ajax applications

Myungjin Lee[†], Sumeet Singh, Ramana Rao Kompella[†]
[†]Purdue University

ABSTRACT

Interactive web applications powered by new technologies such as Asynchronous Javascript and XML (AJAX) have undeniably altered the course of the Web. Despite their popularity, the associated machinery for characterization, measurement, and monitoring of these AJAX-based applications has lagged behind significantly. Part of the problem is that there exists no systematic way to generate AJAX workloads and observe their network behavior today. This paper focuses on addressing this issue by developing a tool, called AjaxTracker, that automatically mimics a human interaction with an AJAX-powered web site and collects associated network traces. These traces can further be post-processed to understand various characteristics of AJAX applications. Given the completely automated nature of our tool, we can study several popular AJAX Web services, such as mail and maps applications from popular providers, both across time as well as space (different vantage points). Our results using a month-long study revealed that AJAX application signatures exhibit remarkable similarity along several features across days. Somewhat surprisingly, there seems to be a lot of similarity across applications that belong to the same category (such as maps, mail) but different providers (such as Google, Yahoo).

1. INTRODUCTION

The advent of a new breed of interactive applications has changed the face of the Web significantly. These include interactive versions of existing Web services such as maps and email, as well as Web-based offerings of traditional desktop applications such as spreadsheets, presentations and word processors. The core of these interactive applications consists of several new technologies such as XHTML, XML, CSS and in particular, Asynchronous Javascript and XML (AJAX) [6].

The continued success of these applications, however, hinges on whether these new technologies can provide the level of interactivity available in the traditional desktop counterparts for these applications. It is important, therefore, for the stake-holders such as customers as well as service providers to continuously study and monitor the characteristics of these applications. An essential ingredient in such an endeavor is a framework for scalable measurement and characterization of these applications.

These interactive applications behave significantly differently from the traditional web. This difference is largely due to the fact that each mouse movement can lead to a transaction between the client and the server, with each transaction involving an exchange of one or many messages. Moreover, such transactions can sometimes happen asynchronously and autonomously even without the user requesting by clicking on an object. For example, the auto-save feature in AJAX-enabled email applications regularly sends a copy of the email message being currently composed to the server. Thus, these applications represent a fundamental shift from the traditional click-to-refresh model that was associated with the Web.

Given their differences with traditional Web services, contemporary measurements studies on Web traffic characteristics do not readily apply in the context of AJAX applications. Similarly, several traffic generators (e.g., [5] and those listed in [4]) that have been proposed before to reproduce network behavior of applications such as Web services do not completely capture the AJAX characteristics. This is in part because we do not completely understand how different AJAX applications behave yet, and partly because they continuously evolve over time. Recently, researchers have conducted some measurement studies to characterize the network-level behavior of AJAX applications [10] using passive network traces. Since they do not control the workload generation, they do not completely capture all (or at least a large sequence of) operating modes of these applications, which we believe is important to comprehensively study and understand these applications.

Our goal in this paper is to focus on two main aspects of AJAX-based Web services. First, we wish to develop a tool that can automatically generate, AJAX application workloads by simulating human like behavior, and collecting measurements that are representative of these new applications. Second, given that they are evolving, we wish to characterize the evolution of these applications over large periods of time.

Our first contribution in this paper is the design of a tool called AjaxTracker that satisfies the twin goals of user emulation of AJAX applications and periodic data collection of these AJAX websites. At the heart of our approach is an autonomous navigation engine that can emulate typical mouse events on a web browser (including events like drag and drop) and capturing network traces during these AJAX sessions. These captured traces are then post-processed to observe characteristics of the AJAX applications. Using AjaxTracker, one can script the interactive navigation process to be followed on a given Web site—just as a human would interact—and then replay the process over several days, and possibly at several locations thus enabling large-scale data collection for these applications. Such a temporal as well as spatial data collection allows us to characterize and observe anomalies and abnormalities, that are otherwise very difficult to observe. We discuss the details of the tool in Section 2

Our second contribution in this paper is the analysis of several representative AJAX applications such as maps and mail services from Yahoo, Google and MSN. In order to study each application session individually, we must consider all packets (possibly belonging to different flows) that are part of the exchange between the client and the Web server(s). Thus, we run the tool at the end host with one application session at a time, so that there is no interference from any other client to server interaction. We characterize each of these representative applications using several features such as request and response message length and inter-request times. Our results indicate that several of these features remain consistent over time. In addition, features such as inter-request times are similar for applications that belong to the same category even across providers, suggesting that similar applications are probably architected in similar fashion. Section 3 discusses these results in more detail.

2. MEASUREMENT COLLECTION

Traffic measurement and characterization is an essential component of various network management tasks such as provisioning, security, and performance monitoring. Much of the existing work on network traffic classification and characterization routinely clubs all HTTP traffic over port 80 into one singular class of traffic encompassed by the WWW. Although, this coarse classification may have been sufficient in the past where each service had a different port and protocol designated for itself, it may not be justified to do so in the current landscape where HTTP is increasingly used as a transport protocol for serving several services and applications —including document editors, spreadsheets, maps, mail and video. Therefore, it is important to further classify HTTP traffic into groups of applications.

The granularity at which one needs to characterize or classify different applications can be quite varied depending on the particular scenario. Given the latest trend towards interactive applications based on AJAX technology, at a minimum, we may want to identify whether Web traffic we observe is AJAX-based or not, and thus, can restrict ourselves to studying the differences between AJAX-enabled and non-AJAX applications. One level of sophistication we may consider is to break up the AJAX traffic into various application categories such as Maps, Word processors, Spreadsheets, etc. A further level of detail may include the provider as well. For example, Google Maps could be considered different from Microsoft Live Search Maps (MS Maps) or Yahoo Local Maps (Yahoo Maps). In this paper, we focus on these fine-granular AJAX applications identified by both the category as well as the provider.

With this application landscape in mind, our goal is to develop a tool that can enable us to automatically collect measurements that can effectively characterize the network level behavior of these applications. Before we describe our tool in more detail, we first motivate the need for a new tool by describing why existing tools do not automatically provide these functions.

2.1 Difficulties of large-scale data collection

One way to measure network-level characteristics of various applications is by employing a packet sniffer that collects packets in the middle of the network (say, at a gateway router). The problem with this approach, however, is the difference between a TCP-level flow (consisting of the 5-tuple) and an application session. An application session (such as when a user navigates Google Maps) may actually result in several TCP sessions because a single service may be provided through different servers, or different parts of a single page are obtained from different servers. Thus, identifying the group of flows that all belong to a single application session can be quite challenging. Besides, this approach only allows us to observe how these AJAX applications are being used in the wild, as opposed to controlled generation of AJAX workloads and observing their behavior.

Our approach, therefore, is to perform application measurements at the end-host. By ensuring that there exists only one application session at any given time, we can collect packet traces that are unique for that particular session, even if the session itself consists of connections to several servers or multiple connections to the same server. Given that we need session characteristics, we need a way to emulate user application sessions. A simple option to automatically generate web traffic workloads is to use web crawlers like wget to periodically download several websites and simultaneously collect traffic traces, which can then be post-processed for characterization. There are, however, several limitations with wget-like crawlers: First, because these applications make extensive use of client-side scripting, one needs to actually execute the scripts in order to obtain the complete context of the web page. Traditional crawlers are tailor made for navigating links embedded within a web page thus they simply parse it to identify the embedded links. Second, traditional crawlers have no built in mechanisms to interact with the interactive controls (like drag-and-drop), which require mouse or keyboard input, and are fairly common in these new applications.

It is, therefore, unclear whether the downloads generated by such web crawlers is representative of the true workloads. Next, we describe the design of our tool which addresses these challenges.

2.2 Design of AjaxTracker

We observe that a full-fledged web browser is required to generate the application sessions, which can subsequently be sniffed on the wire. Thus, our approach revolves around exploiting the web browser to artificially emulate user level clicks and other actions such as drag-and-drop to influence the browser into executing a sequence of user actions in a controlled and flexible fashion.

Algorithm 1 Procedure of AjaxTracker.

```
collect_web_packet_data($url$) begin
run_tcpdump()
run_browser($url$)
wait for 3 seconds till web browser is launched
run_event_generator($scenario\_file$)
wait until event_generator terminates
kill_process($browser$)
$timer$ = 0
while http connection is open and timer $\leq$ 120 do
   wait for 10 seconds
   $timer$ = $timer$ + 10
end while
kill_process($tcpdump$)
```

Our tool, called AjaxTracker, is mainly composed of tcpdump [3], an event generator and a web browser. The event generator, which forms the bulk of our tool, produces the appropriate mouse or keyboard events and provides them to the web browser, which executes the actions, in turn, generating the required set of measurements needed to characterize the particular application. The pseudocode of our process is shown in Algorithm 1. AjaxTracker first runs tcpdump, then launches the web browser, and executes the event generator until all specified events have been processed.

The event generator is a command-line program which is developed using C++, GTK+ and the X library. It accepts several parameters, including the URL to navigate, the web application type (a choice between map, mail, docs and general), a total time duration and a scenario file. The scenario file contains the sequence of events and the position of objects (like link, button, etc.) in a web site. For example, on the Google maps web page, the scenario file could specify several operations such as zooming into a location, entering an address in the search bar and clicking the submit button, dragging the map from one location to another, or some such combination.

The event generator supports two basic navigation modes—static and random. In the static navigation mode, AjaxTracker generates the sequence of events exactly in the order specified in the scenario file. In the random mode, it tries to generate events as randomly as possible, while still following the scenario file as a guideline. For example, the city location can be made random (within a set of cities specified). Optionally, the mouse click can be randomly placed on the map and dragged to random places. Thus, within the application context, the tool allows specification that controls how a given user session looks like. This flexibility is one of the hallmarks of our tool. For applications that require a login—such as mail or Google Docs—the tool performs the login process before proceeding with the event generator. If the tool encounters a generic application, one that does not require a scenario file, then the event generator dynamically generates the events.

Since AjaxTracker uses human specified scenario files, significant changes to the user interface by the application provider may cause the tool to malfunction. Though this limitation may seem serious, observations made by the tool over a period of weeks shows considerable consistency in the results of the static navigation mode, barring a few user interface changes that were easy to modify in the scenario file.

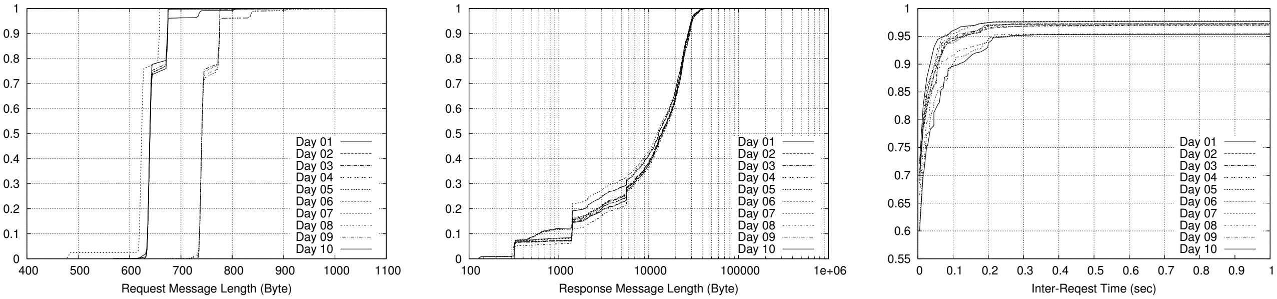
3. MEASUREMENT RESULTS

In this section, we explain our measurement results using AjaxTracker on a sample set of popular AJAX applications that consist of Google Maps, MS Maps, Yahoo Maps, Google Mail, HotMail, Yahoo Mail, and Google Docs.

AjaxTracker considers a set of 5-tuple flows of <*src, dst, src_port, dst_port, protocol*> as a session. Since AjaxTracker aims to characterize mainly AJAX applications, it filters out non-TCP non-Web traffic (i.e., packets that do have port 80 in either the source or destination port fields). We do not need to analyze the packet payload as AjaxTracker only explores one specific AJAX application at a time. Thus, the data collected at the end host does not require any further clustering.

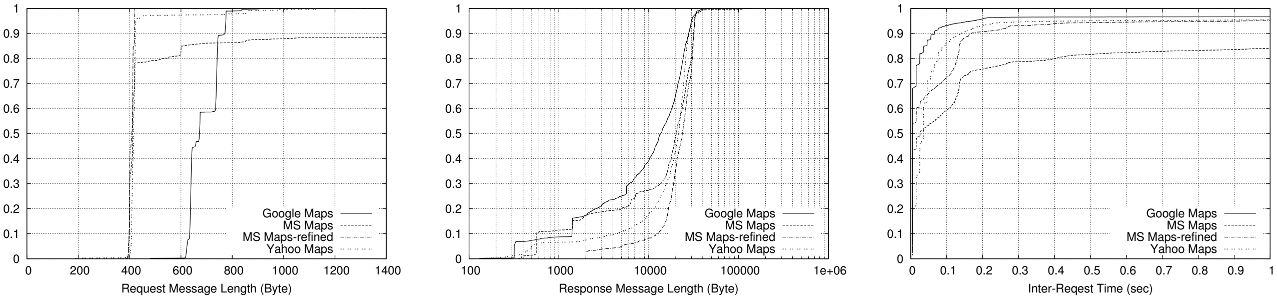
For our measurements, we schedule to run the tool on a daily basis on a Linux based host with a Firefox web-browser. Generally, a web site may contain several documents and images with possibly different sources, in addition to the actual AJAX content. While we can choose to count these as part of overall application session traffic, it might lead to inconsistent behavior either due to differences in the image sizes or other non-AJAX content. So, to eliminate this inconsistency, we exclude flows that have only one transaction, with the rationale being that AJAX-related flows almost always incur more than one transaction.

Once we collected the traces corresponding to individual AJAX sessions, we extracted several features of these sessions in our measurement results. As a first cut, we selected the CDF of request message length (QML), CDF of response message length (RML) and inter-request time (IRT). Note that the list of features we have selected is not exhaustive by any means and could be arbitrarily extended. The features



(a) CDF of request message length.
(b) CDF of response message length.
(c) CDF of inter-request time.

Figure 1: Google Maps' traffic patterns over time.



(a) CDF of request message length.
(b) CDF of response message length.
(c) CDF of inter-request time.

Figure 2: Similarity among map applications.

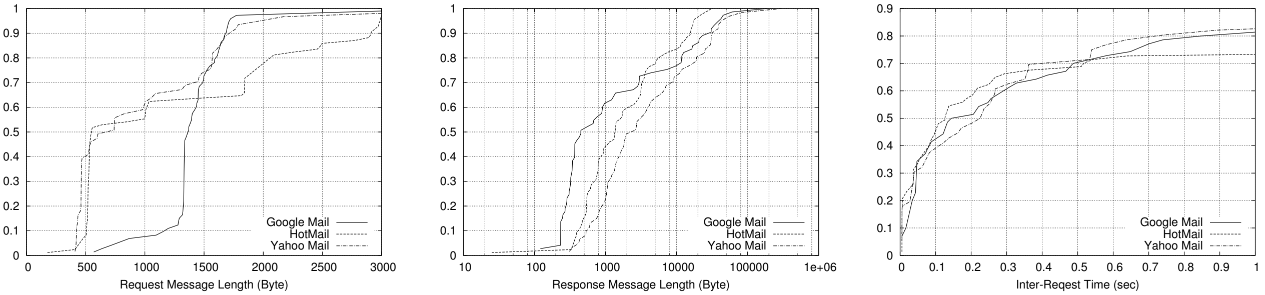
we have selected have less dependence on the location of the vantage point or on the network conditions. For example, since we do flow reassembly, these features do not look at individual packets or their timing, and hence capture more intrinsic characteristics of the AJAX applications.

Given the automated nature of our tool, we could conduct longitudinal study on how these applications vary over time, whether we could see similarity across applications, and how sensitive the measurements are to the exact sequence of navigation steps. We can even repeat the same experiments at different vantage points and compare their differences; we however did not perform this study yet and is part of our future work. We summarize our results and observations based on a month-long deployment of the tool on a university campus machine.

Similarity over time. Our goal in this experiment is to explore whether AJAX applications exhibit temporal similarity. Thus, we configured AjaxTracker to repeat the same set of steps and collect information for the different sessions over a period of 10 days. Figure 1 shows the three features extracted out of Google Maps' traffic. The figure shows that there exists no significant difference over time for all the features except in the QML (in Figure 1(a)). The difference also appears to be just a shift along the x-axis for four out of the 10 days. We believe these changes are the result of developers testing and patching new functionality. Similarly, with Google docs (not shown in the figure), we observed abnormal patterns because of user interface changes. We had to modify the tool to adjust to the new UI by simply rewriting the scenario file. Once we did that, we observed that the QML was consistent.

Similarity within application group. We next group together different providers' version of the same service (such as as Google Maps, Yahoo Maps, and MSN Maps) into one category (shown in Figure 2). In Figure 2(a), we can observe that MS Maps and Yahoo Maps have very similar QML distributions until about the 80 percentile mark. Upon manual inspection, we found that the largest 20% of QMLs of MS Maps belonged to requests for javascript code or requests sent to servers in 2o7.net domain used by Omniture, Inc, which collect any session data of visitors to the customer websites on behalf of the customer (in this case, Microsoft). The curve labeled MS Maps-refined removes these requests and responses from the original data. We can clearly observe that once we prune these requests out, the curves look much similar both for the RML and IRT features.

In Figure 2(c), for all applications, at least 50 percent of IRTs are less than 40ms. Particularly, 92% of Google Maps' IRTs and 85% of Yahoo Maps' IRTs are around 100ms, and thus are quite similar along this feature. Google Maps achieves very fast requests by usually opening up to eight connections and making requests simultaneously. Similarly, Yahoo Maps makes between two to five connections with the same server. In case of MS Maps, a web browser communicates with as

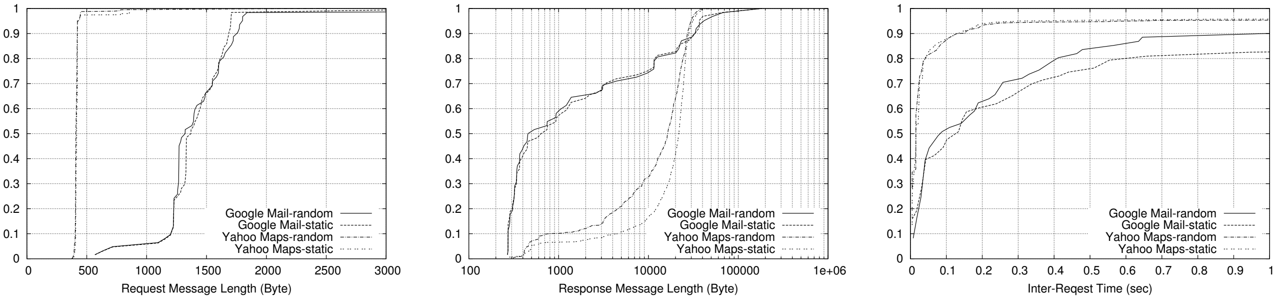


(a) CDF of request message length.

(b) CDF of response message length.

(c) CDF of inter-request time.

Figure 3: Similarity among mail applications.



(a) CDF of request message length.

(b) CDF of response message length.

(c) CDF of inter-request time.

Figure 4: Comparison between static and random navigations.

many as four different servers with up to two connections per server. Note that, for For MS Maps we aggregate the different sessions which download tiles images from four different servers into one session in order to present MS Maps' IRT exactly.

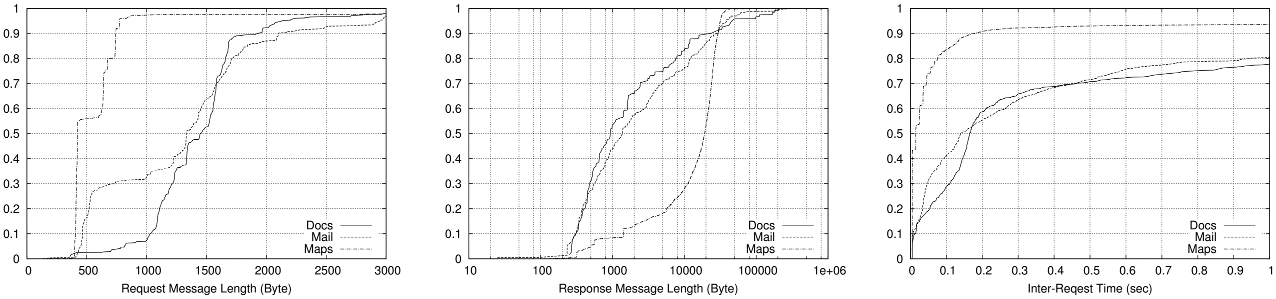
For mail applications (as shown in Figure 3), we find similarity along the IRT feature, which suggests that all the applications use similar prefetching mechanisms. The RML feature also appears to follow a similar trend, while in QML distribution, Google Mail is quite different while Yahoo Mail and HotMail are quite similar. In general, it is hard to expect that the QML distributions are going to be similar because typical requests consist of HTTP headers which may be different for different Web sites even for the same type of service. However, RML to some extent, and IRT to a large extent appear to be quite similar to each other indicating that same category of AJAX applications even across providers may exhibit quite a bit of similarity.

Comparison between Static and Random Navigations. To explore whether these features are tied to specific navigation steps, we compare static and random navigation modes for AjaxTracker across two applications: Google Mail, and Yahoo Maps. Figure 4 shows the similarities between static and random navigations. Google Mail has similar patterns for QML and RML graphs, but has slight difference in IRT graph. For Yahoo Maps, the similarities between static and random navigations are clearly shown in terms of QML and IRT, and the dissimilarity in the RML graph is also tolerable. It can observed, somewhat surprisingly, that AJAX applications have similar features (particularly along the IRT feature) even when the exact navigation steps are quite different.

Dissimilarity across application groups. We mainly focus on differences between different application groups. Figure 5 shows the differences between mail, maps, and docs. For each application group, we consider the aggregate distribution across all providers. We can observe from Figure 5 that the maps application group exhibits quite different characteristics in comparison with both mail and docs. Both mail and docs share a lot of similarity, particularly with respect to the RML and IRT distributions. We believe that this could be because of the fact that the architecture of these applications is quite similar. For example, they both employ similar prefetch characteristics for data outside of the visible window within the AJAX webpage. They both have traffic upload characteristics, when for example, a mail is sent or automatically saved, just as a document could be automatically saved.

4. RELATED WORK

Given the recent trend in the growth in popularity of AJAX based applications, there has been very limited research work either geared towards devising a tool that can collect proper measurements or studying the overall characteristics of these



(a) CDF of request message length.

(b) CDF of response message length.

(c) CDF of inter-request time.

Figure 5: Dissimilarity among application groups.

applications.

A recent paper by Schneider et al. [10] made one of the first attempts to characterize AJAX enabled application protocols by examining the contents of HTTP headers over ISP traces. They extract features that are similar to ours. However, our work is significantly different from theirs, as we focus on generating user interactions with a web page in a controlled fashion to generate our traces, while they rely only on passively obtained data which is quite restrictive. Besides, our analysis includes several new results that have not been reported before in literature.

Several Web traffic generators based on statistical models have contributed to understanding the impact of Web traffic (see [4] for a list of these). These tools, however, do not factor the exact AJAX application traffic characteristics, as there does not exist a model for AJAX application behavior yet. Our tool allows us to conduct large-scale studies on the characteristics of various AJAX applications in the wild, thus contributing to the creation of a model which can then possibly be integrated into the afore-mentioned tools.

Web automation tools such as sahi [1] and SWAT [2] enable one to navigate a web site automatically and repeatedly. However, these tools are mainly tailored for static navigation of websites using automatic filling of forms and lack the functionality to interact with AJAX enabled applications (such as emulate mouse drag-and-drop events).

Roughan et al. [9] leverage statistical information of flow data such as mean and standard variance of packet size, connection duration, packet inter-arrival period, and so on. They then attempt to classify applications by applying machine learning techniques. Ma et al. [8] propose an unsupervised protocol inference framework that employs both payload as well as statistical information to classify packets. Finally, BLINC [7] depends on the social behavior between hosts to identify applications. We can use our work as a mechanism to obtain specific traffic signatures that can assist in devising an AJAX classifier similar to these works and thus is complementary to these approaches.

5. CONCLUSION

As the popularity of AJAX-powered Web services increases, it becomes critical to study and understand their network-level behavior. To this end, we have described AjaxTracker, a tool that is capable of both statically as well as randomly navigating AJAX-based Web sites. Our tool, successfully captures realistic network-level flow measurements by imitating a set of mouse events that in turn leads to an exchange of messages between the client and the Web server. By running AjaxTracker at a stand-alone server and iterating through several web sites, one at a time, we can automatically collect and track several AJAX-enabled Web sites in a scalable fashion. By post-processing the individual trace, we can extract the unique features for the particular application. Our measurement results using data collected over a period of more than one month show that many features for each of the Web sites remain consistent over days. Somewhat surprisingly, we also found that there exist similarities between similar AJAX-based applications (such as maps and mail) offered by different providers, which suggest that there are some fundamental characteristics of these applications that are shared across providers. As part of ongoing work, we are exploring how we can use these features (and possibly new features) to scalably classify AJAX applications in the middle of the network.

6. REFERENCES

[1] Sahi - Web Automation and Test Tool. http://sahi.sourceforge.net/.

[2] Simple Web Automation Tool. http://swat.sourceforge.net/.

[3] tcpdump. http://www.tcpdump.org/.

[4] Traffic Generators for Internet Traffic. http://www.icir.org/models/trafficgenerators.html.

[5] P. Barford and M. E. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of Performance '98/SIGMETRICS '98*, pages 151–160, July 1998.

[6] D. Crane, E. Pascarello, and D. James. *Ajax in Action*. Manning, 2006.

[7] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: Multilevel Traffic Classification in the Dark. *ACM SIGCOMM Computer Communication Review*, 35(4):229–240, October 2005.

[8] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected Means of Protocol Inference. In *ACM SIGCOMM Conference on Internet Measurement*, pages 313–326, October 2006.

[9] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-Service Mapping for QoS: A Statistical Signature-based Approach to IP Traffic Classification. In *ACM SIGCOMM Conference on Internet Measurement*, pages 135–148, October 2004.

[10] F. Schneider, S. Agarwal, T. Alpcan, and A. Feldmann. The New Web: Characterizing AJAX Traffic. In *International Conference on Passive and Active Network Measurement*, April 2008.