IntegraTag: a Framework for High-Fidelity Web Client Measurement

Charles Thomas*
*comScore
cthomas@comscore.com

Jeff Kline*
*comScore
jkline@comscore.com

Paul Barford*†
*comScore
†University of Wisconsin
pbarford@comscore.com

***Abstract*—Collecting information from web clients without explicit input from users is important in a variety of contexts including content customization, experience personalization, accounting and online advertising. A standard approach for gathering web client telemetry is through deployment of Javascript instrumentation that is placed either directly on web pages or through third-party "tags" that are referenced in web pages. In this paper we present a design study of web client measurement methods. The objective of our work is to enhance understanding of best practices in web client measurement toward the goal of developing future tags that are reliable, robust and efficient. We begin by conducting a detailed examination of Javascript instrumentation collected from five well known third party services. Our analysis shows that these code-bases have diverse capabilities and return a broad range of client characteristics. Next, we describe a web client measurement framework and an implementation that we call IntegraTag, which enables us to examine details of performance, accuracy and reliability through live deployments. We use IntegraTag to conduct case studies of tag behavior on a single website which resulted in over 500K page-loads, and on a publisher network which resulted in over 150M page-loads. We establish a lower-bound on the tag's reporting fidelity using a Bernoulli trial. We report on the wide range of client characteristics returned by IntegraTag, as well as its performance and robustness.**

I. Introduction

Web clients are defined broadly by a variety of features including: *(i)* the device type (*e.g.,* desktop, smartphone, tablet, etc.), *(ii)* the software configuration running on the device (*e.g.,* operating system, browser, plugins, etc.), *(iii)* the user of the device (to the extent where it can be inferred), and *(iv)* the environment of use (*e.g.,* location, time of day, etc.).

Accurate identification and reporting of web client characteristics is important in a variety of contexts. Web publishers can tune content and user experience based on this information. For example, the manner in which a page renders can depend on device type, the physical orientation of the device and browser configuration. Similarly, marketers and advertisers use a variety of web client characteristics to aid in the selection of ads that are delivered to clients. Web tracking and accounting services such as Google Analytics [1] gather and report web client characteristics to enable publishers to understand their audience. Finally, technically savvy fraudsters aim to profit by deceiving reporting mechanisms [25]. This impacts both marketers and publishers. Accurate reporting is critical for fraud detection applications.

A standard approach for automated collection of web client data (*i.e.,* telemetry) is through Javascript instrumentation, which is generally refered to as a "tag". Javascript tags employ several common techniques (*e.g.* attaching 1x1 tracking pixels to the document, or asynchronous AJAX-style server calls) to report their measurements to the appropriate parties. When the instrumentation is deployed directly by a publisher it is referred to as a first-party deployment. Tags that are coupled with advertisements or otherwise embedded in a web page by someone other than the page's owner are called third-party deployments. Web client telemetry is frequently collected by third parties such as advertisers and data aggregators like Oracle/BlueKai [4] who work in partnership with publishers.

The primary challenges in accurate collection of web client telemetry stem from the diversity of devices, browsers and security policies that are in current use. Device and browser combination often impact how Javascript executes, what methods one may call and the values that are returned. This complicates development and debugging of broadly-compatible instrumentation. Further, placement-specific security restrictions, such as same-origin policies or HTML5 sandboxing limit the actions that a Javascript tag may successfully execute. Finally, JavaScript tags rarely allow a user to provide feedback to the tag developer. When a Javascript tag fails to properly execute, both user and developer may be unaware of the failure. This highlights the need to integrate a robust "tag health" monitoring system into the tag's functionality.

In this paper, we present a design study of tag-based methods for collecting web client data. The goal of our work is to elucidate issues that are central to development of tags that are flexible, robust and efficient. Our focus is on Javascript-based, third-party tags but our findings are applicable to first-party and non-Javascript instrumentation (*e.g.,* some tags are deployed in Flash).

We begin by presenting an "insiders view" of Javascript instrumentation that is delivered by five different widely deployed third-party tags. Our objective is to assess current best practices in web client measurement. Although gathering the code-bases is trivial (JavaScript is transmitted in plain text), understanding the code's function and purpose can be a time-intensive exercise. The code is often deliberately obfuscated to thwart reverse-engineering. Due to the rise in blacklist-based ad-blocking, there is also incentive for some deployments to host their code on little-known domain names. JavaScript

978-0-9883045-1-2/16 $31.00 © 2016 ITC

code may also be mangled by minimization tools. The same code-base can execute along different paths depending on the deployment scenario and the particular state of the client that executes it. Through careful manual analysis, our results show that third-party tags are designed to return a wide variety of web client telemetry on aspects such as browser configuration, screen configuration, web page content, user actions and environmental variables to name a few.

We complement our static analysis of web tags with experiments in live deployments that are aimed at understanding basic issues of tag behavior and client diversity. To conduct these experiments we developed *IntegraTag*. The high-level design objective of IntegraTag is to assess and diagnose tag compatibility, reliability and performance as well as have the ability to return information about a wide range of web client features in live deployments. We implemented IntegraTag in Javascript using a design that enables features to be flexibly enabled for different diagnostics and performance tests.

We conducted a case study using IntegraTag in two live deployments. The first deployment ran the tag alongside online advertisements that are delivered to a specific live web site over the period of a week. During that period, IntegraTag was activated over half a million times by between 5,000 and 20,000 unique users per day (as identified by cookies placed in accordance with do not track). The second deployment ran on a broader selection of online ads that appeared on 152 different websites resulting in over 150M activations of the tag over a period of four days.

We configured IntegraTag to request a *ping pixel* on 1% of page loads. For greater reliability, the ping request was initiated prior to executing more complex tag features. Additionally, the pixel request only reported information that can be gathered without causing fatal errors. This included a placement identifier (*e.g.* domain name or a URL), a unique user identifier, a session identifier and a cache-breaker (a string of random text appended to a URL to prevent a browser from caching the URL). Since the ping was designed to be loaded first and have minimal functionality, we believed ping volume would be at least 1% of the total volume measured by the more complex environment-probing code. Surprisingly, the number of observed pings was significantly *below* the number that theory predicted. The discrepancy sets a lower bound on the fidelity of our measurements.

The second aspect of our case study considered the diverse characteristics of web clients. Specifically, we report on the results of data collected by IntegraTag configured to collect 47 different client characteristics. Our results highlight the diversity of client characteristics that are likely to be of interest to third party tagging entities. We provide examples of specific characteristics that were gathered to highlight the diversity of client platforms. We examine the impact of IntegraTag on page load times. We find that its 32K code-base typically completes its browser probing activities in under 50ms. This illustrates that rich telemetry can be acquired and reported in a manner that does not interfere with overall user experience, which is a key concern for publishers and by extension third party measurement entities.

Finally, an important consideration in web client instrumentation is user privacy, which has been discussed in the popular press (*e.g.,* [23]) and research literature (*e.g.,* [12], [17], [15]). We do not address the issue of privacy specifically in this paper. It goes without saying that all major entities that deploy tags are concerned with legal compliance and user perception. By providing the insiders view and reporting on the details of methodology for web client instrumentation, we believe that our work helps to elucidate that conversation about privacy and could lead to new methods that strike a more widely accepted balance between privacy and functionality. We also believe that our work provides a roadmap for further investigation of these important issues.

The remainder of this paper is organized as follows. In Section II we provide an overview of tags. In Section III, we report on results of a study of a selection of third-party instrumentation that is broadly deployed on web pages. In Section IV, we describe the design and implementation of IntegraTag. The results of our deployment of IntegraTag on live web pages are reported in Section V. We discuss prior studies that inform our work in Section VI. We summarize, conclude and discuss future work in Section VII.

II. Web Tag Basics

We begin with a general discussion about practical aspects of instrumenting web pages with JavaScript tags. While the underlying technologies are well-known and the individual techniques straightforward, challenges arise from non-standard browser behaviors, deployment errors, unexpected user behaviors and bugs in the instrumentation itself. We limit our attention to JavaScript-based tags with brief comments on other techniques towards the end.

Information reported by JavaScript tags is currently the major tool by which online media companies (publishers, brands, agencies) measure their audiences. Since this code regularly executes alongside content that has significant monetary value, tags are typically designed in accordance with the principle that they must reliably execute without interfering with a user's overall experience. A poor implementation risks shortening mobile device battery life, interfering with a publisher's content or consuming a large volume of web traffic.

A. Tag Deployment

Tags are deployed when a web publisher or a brand that is running an ad campaign desires to measure some feature of their traffic that they are unable to derive from their own logs. They often address this by partnering with a third-party telemetry service. This service generates a customized block of code or a URL that points to a pixel/image or an HTTP 204 (No Content) resource hosted on one of its servers or in a CDN. Both types of requests are commonly called "pixel requests". The publisher adds this block of code to all pages that it serves. Ideally, each web browser request issued to the publisher also results in a subsequent request to the telemetry service.

Tag deployments often require careful placement within the page HTML structure. Some tags are designed to be loaded as soon as possible, others must execute as late as possible. Still other tags need to be embedded as either a sibling or child of particular DOM element to function correctly. Incorrect deployment may cause telemetry errors or complete failure in tag execution. Usually a person is required to do this, often in consultation with the telemetry company's support staff. The scope and complexity of tag deployments on large websites has led to development of tag management systems such as Google Tag Manager [1].

Coarse statistics about a publisher's server traffic such as hourly or daily volume are sometimes made available from the publisher's logs. This type of information is often sufficient to identify deployment errors. But a more detailed study of a publisher's logs, however, is rare. This is due in part to the lack of a well-defined key to perform a JOIN across different data sets. Even if all parties are setting cookies and the web client is storing them, cross-domain communication is typically prohibited. As a result, cookie syncing is usually not possible unless a dedicated process has been undertaken. User privacy is also a major concern. This is especially relevant where account logins are required or the site concerns personal, financial or medical information.

B. Data Types

The information that is transmitted in JavaScript tag-based telemetry is diverse. When third party services are used, telemetry includes an account identifier. For ad campaigns, the characteristics of an audience's response to individual ad creatives (the image or video that a user sees as the advertisement) or placements is often of interest. As a result, ad creative identifiers are often included in ad campaign telemetry.

Sometimes individual impressions and page views are associated with a unique identifier. This is useful, if say a publisher wishes to measure dwell time on a page. This could, in principle, be measured with a monolithic pixel request that fires after a set time interval or when the page unloads. But for this approach to work, one requires consistent behavior associated with several unrelated page events including page unload, window close, tab close, process quit and page reload.

An alternative design relies on multiple pixel requests. This requires establishing the notion of an atomic event, *e.g.,* a page view, a widget load or an impression and associating this event a key. Subsequent processing then must perform a JOIN on the several streams of data.

The context in which the tag executes can change how the tag behaves and strongly affects what it may measure. For example, a tag that executes on the top-level frame of the window can often collect a range of attributes and make observations about user actions. In contrast, a tag that lives within a sandboxed iframe will typically have access to relatively little information about user behavior. This has a direct impact on the kind of reporting or analysis that is possible on a deployment.

C. Data Reporting Methods

Most Javascript tag data is reported via query strings. To be more precise, telemetry is reported via HTTP GET requests to a 1x1 "tracking pixel". The basic idea is that the Javascript from the tag runs and actively gathers as much information as necessary about the client browser and the viewing environment. This information is packaged (in some cases distilled, aggregated, and/or encoded) as a set of parameters and associated values. At this point, the Javascript requests an image from a the tagging entity's server. The image request usually returns a 1x1 pixel image which is attached to the document where the tag was placed. Telemetry is encoded and transmitted in the URL used to request the image. For example:

```
http://foo.com/p?parm1=val1&parm2=val2&parm3=val3...
```

where "p" is the name of the image file being requested. The server ignores the list of telemetry parameters (the string after the "?" character) and simply returns the image file to the browser. However, the entire request, including the parameters, is recorded in the server logs along with a date/timestamp, the User Agent string, the IP address of the client that made the request, and several other bits of information about the transaction. In this way, the telemetry is available for subsequent processing.

If the information in the telemetry can be conveyed in under about 2000[2] characters then this method is sufficient. But ad serving commonly involves many different entities working to serve a single impression. It is not uncommon to include the full referrer URL and additional query string parameters in requests, so it can happen that browsers fail to send the request or requests are truncated due to excessively long (but otherwise legitimate) query strings.

To work around any URL length upper bounds, an alternative technique is to include telemetry as part of an HTTP POST event. Indeed, two of the tags we examined (and the IntegraTag itself) return telemetry using this POST method. POST requests have several features which make them attractive for returning telemetry: *(i)* POST requests are "memoryless" in the sense that they are not cached and they do not remain in the browser history, *(ii)* POST requests always attempt to travel to the server where they will be logged, and *(iii)* POST requests have no restrictions on data length. Item *(iii)* is important since it's trivial to generate telemetry strings which are so long they will potentially be truncated or rejected by either the browser or the server from which the pixel is being requested.

This technique was occasionally used to return raw HTML code of the page that hosted the tag. This raises privacy and security concerns. Additionally, if the message body is too

[1] https://www.google.com/analytics/tag-manager/

[2] The character limit for a URL in Microsoft browsers is 2083 characters. See https://support.microsoft.com/en-us/kb/208427.

large, it will interfere with user experience and also potentially incurr large serving costs. Nevertheless, this method has proved invaluable to diagnose sanity check violations and to inform investigations involving fraud where other, less-invasive techniques are uninformative.

A modest commercial deployment may ingest daily volume of between 100 million and 1 billion 1 kilobyte records and have a total daily volume of up to 1 terabyte of raw data. Much of the data that forms the basis of the present paper came from such an environment. Data processing typically is managed with MapReduce frameworks, SQL-based solutions and other tools. A common hardware configuration is either based at commercial data centers or with custom-built solutions.

D. Other Techniques

Mobile application telemetry is commonly implemented as a combination of compiled code and JavaScript. The compiled code may be Java or Objective C and the binaries often have permission to read device attributes that Javascript alone cannot access. In particular, it is common to report a device's IDFA (for iOS-based devices) or AAID (for Android devices) in commercial web traffic.

Other types of identifiers that attempt to associate each record with an individual browser or device have also been reported. Although these non-standard approaches such as browser fingerprinting, flash super-cookies and ISP cookie injection are known to have been employed, all have been met with resistance in part because users are either unable to opt-out of these programs or were enrolled without consent.

We close this section by referencing Table 1. This table will be discussed in greater length below but we introduce it here since it effectively highlights the diversity of tag functionality that exists among commercial tags[3].

III. Characteristics of Canonical Web Tags

We begin our study of instrumentation for user data telemetry by examining tags and associated Javascript from several well-known third party collection entities. Our selection of the set of tags for this study is arbitrary and represents a small subset of all of the tracking entities that are active in the web today. We focused on this set of tags due to their mix of broad deployment, the set of attributes measured which relate to browser settings and our ability to convert the code into a tractable form. Based on our experience, we believe that the general characteristics of this set of tags is representative of what can be found more broadly in the web today.

Tag discovery is a straightforward task: simply crawl a set of pages where one thinks tags may be deployed, and then scan the HTML source for links to the tag source. Indeed, this process can be automated using web crawling methods such as those defined in [24], [5], though a manual crawl is also an effective discovery method of widely-deployed tags. To keep the Javascript code as compact as possible, many of the tags that we found have run the final source code through a compressor (*e.g.,* Minify JS [2]), which strips out whitespace and minimizes variable and function names. For example, a call to

$$getSmartStringHash(mime\_array, encoding\_type)$$

might become simply *a(x,y)* after compression. The result is not easily readable by a human. There are tools available which can streamline the process of analyzing a compressed tag (*e.g.,* JSNice [13]). However, it is much less effort to start with a uncompressed version of the tag.

Once a tag is identified and downloaded, analysis of the Javascript instrumentation can be performed. While Javascript can be examined directly, instrumentation code from different entities uses a variety of conventions and paradigms to implement different functions. Some tags use obfuscation or even encryption [4], which we hypothesize is done to reduce the human-readability of the code, obfuscate business partnerships or possibly to escape notice from automated analysis tools. Another challenge is that some companies deploy multiple tags, each tag designed for a different task. For example, a tag for presenting survey options might look similar to, and might deployed almost identically to, a tag from the same company that performs analytic tasks. Thus, identifying the general objectives of tagging entities can require examination of a variety of code bases.

For the purposes of this study, we narrowed our focus to the functions that gathered client metrics, and did not attempt to document every function that the third party tags performed. Many tags, for example, have varied (and in some cases, redundant) methods of communicating results back to their data-collection servers. Some tags, in addition to their analytic capabilities, also serve advertisements or other data content to the page on which they are deployed. We leave detailed examination of those functions for future work.

We selected five third-party tags for our study. We refer to the tags by the major function of the entity that has deployed them: Data Aggregator, Search Engine, News Service, Ratings Provider and Ad Metrics Service. In each case, the tags consist of a small piece of Javascript code that calls for the larger body of Javascript instrumentation that is the focus of our analysis. The instrumentation code bases that we examined varied between 15KB and 72KB.

Our analysis highlights common and distinct functionality in each of the tags that we examined. Space limitations preclude detailed descriptions of all functions that we found in the tags so we group functions into major categories as can be seen in Table 1. The table shows that most of the tags attempt to acquire information about the local context in which the code is executed. Examples include the user's screen size, browser type, installed major plugins (*e.g.,* Javascript and Flash), and the local timezone.

Our analysis also reveals varying degrees of analytical depth with regard to browser, screen, window, component, document,

[3]The names of the companies listed in this table and the details of their code's functionality are obfuscated based on advice from council.

[4]The Ad Metrics Services tag applies `rot13` encryption to one of its internal data structures.An unrelated video tag (not discussed here) employs the `crypto.js` tag to use strong encryption as part of video ad delivery.

Category	IntegraTag	Data Agg.	Search Eng.	News Service	Ratings Co.	Ad Metrics Co.
Browser features	10	8	3	1	2	1
Screen metrics	5	5	3	2	3	-
Window metrics	5	2	2	1	-	3
User actions	5	-	2	1	1	2
Plugins	4	3	2	-	1	2
Document attr.	2	2	2	2	1	1
Date & time metric	2	3	-	1	1	1
navigator.platform	1	1	1	-	-	-
navigator.cookieEnabled	1	1	-	-	1	-
navigator.userAgent	1	1	1	-	-	1
navigator.doNotTrack	1	1	-	-	-	-
Supp. MIME types	1	1	-	-	-	-
Mobile detection	1	1	-	-	-	-

Fig. 1. The above table summarizes the functionality of six tags. Rather than individually listing all attributes measured by each tag, we have aggregated related attributes into labeled bins. The integers in the table reflect the bin's size. For example if a tag measures a web browser's locale setting, its time zone and the reported date, these measurements are binned together in "Date & time metric" and the value in the tag's column is 3. The left-hand column describes the bins, the column headers describe the tag.

and user action metrics. Some of these metrics are acquired simply. For example, once it is determined to exist, one can inspect the built in attribute *navigator.language* in most browsers to obtain a string describing the default language of the browser:

```
if (typeof navigator !== "undefined" &&
    typeof navigator.language !== "undefined")
{
  value = navigator.language;
}
```

Some metrics are much more challenging to acquire. For example, determining if and where the user has clicked on the page involves calls to determine the browser type, calls to install various browser-specific event listeners, a cascade of methods that each try to obtain the click information (each failing to the next method if unsuccessful), and calls to remove the event listeners when the analysis is finished.

Each third-party tag we examined is tailored for the specific needs of the entity which created it. Thus, the presence or absence of any particular function does not imply any kind of deficiency in the instrumentation. For example, the Ad Metrics Service tag seems to have been designed to be paired with a specific ad. As a result much of its functionality involves monitoring interactions with the ad it is shepherding. The tag from the News Service, on the other hand, appears to have functions to facilitate eCommerce; it tracks and reports attributes like order number, tax, shipping, SKU, and promotion codes. The tag from the Data Aggregator, as another example, can do HTML parsing of the page on which the tag is deployed in order to gather contextual information and keywords from the sites where their tags are running.

IV. Designing a Web Tag Framework

The design goal for IntegraTag is an implementation that is *flexible*, *robust* and *efficient*. Flexible recognizes diverse measurement needs and refers to the ability to easily install new or modified data gathering capability in the tag. Robust recognizes the diversity of client configurations and refers to the ability to report any failures that may occur prior to completing execution. This is critical in assessing the veracity of returned telemetry. Efficiency refers to limiting the processing impact on client systems through careful functional implementation.

Since the tag functionality is nontrivial and the integrity of the telemetry is essential to the accuracy of analysis models that rely on the data, we sought to establish ground truth of traffic volume by firing a "ping" pixel with probability p[5]. The ping is designed as the minimum unit of instrumentation so as to avoid platform compatibility issues. It does not call any methods which could potentially result in errors; it simply returns the placement identifier (*e.g.* the domain or URL) and some other minimal diagnostic information. Given the set of pings, it ought to be possible to estimate the overall expected volume of the data. This was a unique feature among the tags included in our survey.

The IntegraTag was designed to gather a wide range of user and environmental information. This includes some information which is typically regarded as redundant or irrelevant to serving ads. As one example, a web client's underlying OS and platform can often be derived from the User Agent. We measured this attribute as reported by Javascript to identify web clients who modified the default User Agent string. A subset of the information returned is highlighted in Figure 1.

Due to the large number of tests the tag must run and the many attributes the tag must attempt to gather, the tag code needs to be both flexible and resilient to fatal errors. During our tests on a subset of our traffic, the typical 24-hour period saw the Javascript code executed 100 million times by dozens of different browser types running inside thousands of

[5]Conservative use of pixels is important since there is a cost associated with serving them at large scale.

TABLE I
USER CHARACTERISTICS REPORTED BY THE INTEGRATAG FROM DEPLOYMENT ON TARGET WEBSITE OVER A ONE WEEK PERIOD.

Day	Users	MIME Confgs.	Browsers	Screen Confgs.	Time Format	Platforms	Languages
4/30/2014	20,646	4,695	3,571	1,652	142	22	39
5/1/2014	12,880	2,375	2,343	1,145	103	19	23
5/2/2014	14,879	2,788	2,615	1,343	103	20	21
5/3/2014	6,312	1,191	1,309	731	88	17	20
5/4/2014	5,958	1,102	1,266	715	77	18	17
5/5/2014	18,392	2,074	2,262	1,154	76	18	17
5/6/2014	20,567	2,303	2,484	1,237	86	19	19

different device configurations. Complete failure of the tag can result if the code tries to carelessly call functions which are not supported by a specific browser version. The IntegraTag is built to be robust to browser compatibility issues: when possible, we probe to ascertain the existence of environmental parameters prior to querying for that information. Each probe is isolated to avoid affecting subsequent probes.

The tag also has the ability to report back to our servers if it was unable to complete one of its functions. If an error occurs at any of forty-seven separate checkpoints during code operation, an error report is generated. If a nonfatal error is encountered, the code will continue to run. It is possible that a single ad impression could generate many error reports. Using the error reports, we have been able to refine specific functions that were shown to be problematic, and reduce the number of errors that are generated by tag operations to less than 1% of impressions. Going forward we can monitor these errors and continue to enhance IntegraTag to more successfully examine the web ecosystem as it changes over time.

We typically reported at least two types of pixel each time the tag executed. The reporting mechanism is compartmentalized so that different types of data are reported independent of each other. Environment parameters (*e.g.* user settings, browser attributes, installed plugins, etc) are gathered at tag launch and are reported immediately via a (environment) pixel. User actions (*e.g.* mouse and keyboard activity, page dwell time, etc) are gathered over a period of time and reported after some delay via a (user) pixel. Failure of the environment tag does not halt firing of the user pixel. Likewise, ping and error telemetry are isolated from all other types of reporting and fired as soon as possible.

IntegraTag is implemented in Javascript. It is written in a highly modular fashion that enables sections of code to be easily enabled/disabled depending on the intended use case.

V. CASE STUDIES FROM INTEGRATAG DEPLOYMENT

We deployed the IntegraTag on ad campaigns that ran on over 150 different websites and collected data during two tests lasting four days and a week, respectively. During each test we recorded impressions, errors and pings along with associated client data. Anomalies in ping and error data triggered several investigations which are summarized below. We start with some illustrative examples to highlight the richness and diversity of client data that was returned by IntegraTag.

The client data is unusually diverse in two ways. First, as already noted, the tag collects a wide variety of browser attributes and environmental settings. Second, for many attributes, the tag does not attempt to normalize or modify the browser response. Instead, the tag simply reports the literal responses of the browser. The result of this is a sometimes surprising range of values. Reporting literal values incurred little extra cost in transmission, storage or processing yet afforded great flexibility for exploratory projects.

TABLE II
ENVIRONMENT PARAMETERS MEASURED OVER A ONE WEEK PERIOD FOR ONE SITE.

Top 10 Language Settings	Count
en-US	317,112
N/A	138,588
en-us	91,143
en	5,679
en-GB	379
en_US	125
es-es	88
en-gb	43
en-CA	43
es-419	41

Table I shows some of the breadth of attributes probed and the wide range of values observed across the seven-day testing period. Even at the level of detail in the table, we are able to derive some insight about the population characteristics: the user population on Wednesday 4/30 is nearly the same as the user population on Tuesday 5/6. But by all other measures, the Wednesday population was significantly more diverse. Since the deployed code base and the placement on the site were held fixed during this period, this feature's cause is external to the tag deployment scenario. Based on experience, features similar to this have been traced to several different causes:

1) A special interest story attracted a small but unusually diverse set of users
2) The web site owner changed its traffic generation partners during this period
3) A browser update occurred during this time period that had both wide reach and altered our tag's functionality

A full investigation into the underlying cause is an expensive endeavor and typically only occurs in cases where fraud, a bug in the code-base or a deployment error is a concern.

TABLE III
IMPRESSION COUNT AND PING PIXEL COUNT GENERATED BY INTEGRATAG ON THE TESTED SITES OVER THE TESTING PERIOD.

Day	impressions	ping count	$\frac{\text{ping count}}{\text{impressions}}$
4/2/2014	43,896,149	433,905	0.988%
4/3/2014	35,095,942	344,691	0.982%
4/4/2014	37,808,414	371,028	0.981%
4/5/2014	46,584,616	463,714	0.995%

Table II displays the most frequently observed results of the browser attribute, *navigator.language*. The table shows seven different ways browsers reported a user preference for English.

In the test deployment, the ping pixel was set to fire at a rate of 1%. Thus, we expected to model the pings as a Bernoulli process with success rate $p = 0.01$. Since the ping was designed to fire more reliably than the environment pixel (which we register as an *impression*), we expected the number of observed pings to be *larger* than p times the number of impressions. Table III shows that the observed ping volume is consistently *less* than 1%. A simple calculation[6] suggests that the observed discrepancy is either an extraordinarily rare event or, more likely in our view, it is an artifact of our measurement apparatus. This simple model establishes a lower-bound on our tag's fidelity.

We have localized the cause of some of these issues to a small number of sites, some of which generate significantly more impressions than pings (*e.g.,* simply by running the tag code repeatedly). Previous experiences with similar site-specific anomalies identified server configuration settings as the cause. But the cause of this ping anomaly is unknown. This result highlights the unexpected and complex behaviors that can occur in the web/advertising eco-system.

The vast majority (over 99%) of error pixels that were generated during our tests (less than 1% of total impressions) were either site-specific or browser-specific. A small number of sites were responsible for the majority of site-specific errors. Three major causes were identified, each due to an action attempted by the tag. The actions are: 1) modify the local document, usually by appending an invisible tracking pixel to the DOM tree, 2) violate a security restriction such as a cross-domain request from within an iframe or 3) violation of a self-imposed limit such as dynamically creating a string whose length exceeds a fixed number of characters. Since multiple error messages may be dispatched on a single page-load, we receive a large number of these notifications.

Browser-specific errors were typically straightforward to identify. For example, users with older versions of Internet Explorer can generate errors because their browsers don't support query functions that are available in newer browsers.

Finally, we examined the execution time of IntegraTag. Publishers and advertisers are sensitive about performance impact of tags since poor performance can degrade user experience and potentially limit the telemetry returned. For instances where IntegraTag ran to completion (*i.e.,* all expected telemetry was returned), the average response time was less than 50ms.

[6]Under the Bernoulli model, a set of $n = 40$M impressions that had pings fire at a rate of $p = 1\%$ results in an expected 400K pings with a standard deviation of $\sqrt{np(1-p)} \approx 629$. Our observed discrepancy is about 10 times this figure – and in the wrong direction.

VI. RELATED WORK

Gathering web client telemetry through various means has a long history that can be traced back to studies in the mid 1990's. One of the first examples was work by Cunha *et al.*, which deployed instrumentation in Mosaic browsers to collect client browsing data [7].

One of the most common forms of web client instrumentation are tools designed to be installed directly on web browsers. These are typically used to monitor a variety of client and network characteristics and often employed to help debug performance problems. Commercial examples of such tools include Firebug [19], neatest [11] and Compuware AJAX Edition [6], which are all Javascript tools focused on assessing page performance characteristics (*e.g.,* latency and throughput). Similarly, Dhawan *et al.* developed the Fathom tool as a Javascript-based, Firefox extension to enable performance measurements from remote clients [8]. Netalyzr is Java app that runs in a client browser and provides a detailed analysis of network connectivity [14]. Another form of instrumentation are systems designed as independent platforms for web client measurement. An excellent example of this is the Webpagetest platform, which provides detailed client performance reports from a selection of browser types and dedicated hosts distributed around the world [26]. While these tools and systems all inform our work, to the best of our knowledge, ours is the first paper to analyze details of third party, tag-based web client instrumentation, which is perhaps the most widely used form of web client measurement in the Internet today.

Web tags have been examined in prior studies that focus on user privacy. The term "web bug" has been used in association with 1x1 tracking pixels, and the term "tracker" is commonly associated with third party code that collects user information. In [16], Martin *et al.* consider the prevalence of web bugs on popular web sites and discuss implications for privacy. Two relatively recent surveys of issues surrounding user tracking and privacy are provided in [17], [15]. Both of these studies highlight the tussle between privacy and online advertising that benefits greatly from user information. Concerns over user tracking have resulted in development of the Ghostery plugin for the Firefox browser, which enables users to see who is tracking them [20]. Many other such mechanisms are available. A taxonomy of different trackers including first and third party as well as social media trackers is described in [24], [10],[18]. By simulating users (based on AOL query search logs), the authors are able to project tracking prevalence and they propose methods to protect user privacy. Similarly, the authors in [12] examine privacy issues related to online advertising activities, and develop a client plugin called Privad [21] that provides targeted ads to users in a privacy preserving fashion. Several recent studies have also considered alternative

methods of user tracking. For example, studies by Eckersley, and Acar *et al.* report on the utility of browser fingerprinting, canvas fingerprinting and evercookies as a means for user tracking [9], [3]. These studies have important implications for privacy but do not address the general issues of client measurement that are the focus of our work. Finally, the work by Nikiforakis *et al.* examines the trust relationships between web sites and remote Javascript library providers (including Javascript that is downloaded by tags) [22]. That study complements our by highlighting vulnerabilities that can emerge due to reliance on remote libraries.

VII. Summary and Conclusions

In this paper we present a study of instrumentation for collecting web client data. Our particular perspective is from third parties that use Javascript instrumentation, which is delivered primarily via embedded tags on web pages. We begin by presenting results from an examination of five instrumentation code-bases from large third party data collection entities. Our examination is focused on understanding the objectives for and methods of user information tracking. Our analysis reveals a wide variety of functions and features in the different code-bases. We find that the instrumentation is focused on characteristics such as browser configuration, screen configuration, web page content, user actions and environmental variables. We also find that each code-base has unique capabilities such as HTML parsing, online commerce tracking, and the ability to be paired with a specific advertisement for a more in-depth analysis of the interactions with that ad.

To understand dynamics of tag execution and the diversity of client characteristics we developed IntegraTag. IntegraTag is a Javascript framework designed to run efficiently, to enable flexible transmission of telemetry and to enable detailed diagnostics of tag behavior. We use IntegraTag to conduct a case study of tag behavior through live deployments. We report on the diversity of characteristics observed in the client population and on unexpected behaviors the highlight the difficulties in web client data gathering. The discrepancy between observation and theory of a Bernoulli trial establishes a lower-bound on IntegraTag's reporting fidelity.

In on-going work we continue refine the capability of IntegraTag to return telemetry in a reliable and efficient fashion. We also plan to pursue development of methods for accomplishing the goals of web client data collection in a privacy preserving fashion.

Acknowledgments

The authors would like to thank Kevin Springborn for his input on early versions of this work.

References

[1] Google Analytics. https://www.google.com/analytics/, 2014.
[2] Minify JS. http://www.minifyjs.com, 2014.
[3] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, AZ, November 2014.
[4] Bluekai. Big Data for Marketing. http://www.bluekai.com, 2014.
[5] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan. An Empirical Study of Web Cookies. In *Proceedings of the World Wide Web Conference (WWW)*, Montreal, Canada, April 2016.
[6] Compuware. Compuware AJAX Edition. https://compuware.com, 2014.
[7] C. Cunha, A. Bestavors, and M. Crovella. Characteristics of WWW Client-based Traces. In *Boston University Technical Report TR-95-010*, Boston, MA, July 1995.
[8] M. Dhawan, J. Samuel, R. Tiexeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson. Fathom: A Browser-based Network Measurement Platform. In *Proceedings of the ACM Internet Measurement Conference*, Boston, MA, November 2012.
[9] P. Eckersley. How Unique Is Your Web Browser? In *Proceedings of PETS'10 Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, 2010.
[10] Marjan Falahrastegar, Hamed Haddadi, Steve Uhlig, and Richard Mortier. Anatomy of the third-party web tracking ecosystem. *CoRR*, abs/1409.1066, 2014.
[11] Google. neatest. https://code.google.com/p/nettest/, 2014.
[12] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the USENIX Network Systems and Design Conference*, Boston, MA, March 2011.
[13] JS Nice. http://www.jsnice.org, 2015.
[14] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: Illuminating The Edge Network. In *Proceedings of the ACM Internet Measurement Conference)*, Melbourne, Australia, November 2010.
[15] B. Krishnamurthy. Pivacy and Online Social Networks: Can Colorless Green Ideas Sleep Furiously? In *Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2013.
[16] D. Martin, H. Wu, and A. Alsaid. Hidden Surveillance by Web Sites: Web Bugs in Contemporary Use. *Communication of the ACM*, 46(12), December 2003.
[17] J. Mayer and J. Mitchell. Third-Party Web Tracking: Policy and Technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2012.
[18] Hassan Metwalley, Stefano Traverso, Marco Mellia, Stanislav Miskovic, and Mario Baldi. *The Online Tracking Horde: A View from Passive Measurements*, pages 111–125. Springer International Publishing, Cham, 2015.
[19] Mozilla. Firebug. http://www.getfirebug.com/, 2014.
[20] Mozilla. Ghostery Plugin. https://addons.mozilla.org/en-US/firefox/addon/ghostery/, 2014.
[21] Mozilla. Privad client. https://addons.mozilla.org/en-US/firefox/addon/privad-client/, 2014.
[22] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. VanAcker, W. Joosen, C. Kruegel, R. Piessens, and G. Vigna. You Are What You Include: Large-scale Evaluation of Remote Javascript Inclusions. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, Raleigh, NC, October 2012.
[23] S. Olsen. Nearly Undetectable Tracking Device Raises Concern. http://news.cnet.com/2100-1017-243077.html, July 2012.
[24] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-party Tracking on the Web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, April 2012.
[25] K. Springborn and P. Barford. Impression Fraud in Online Advertising via Pay-Per-View Networks. In *Proceedings of the 22nd USENIX Security Symposium*, Washington, D.C., August 2013.
[26] Webpagetest. https://www.webpagetest.org, 2014.