ISP-Enabled Behavioral Ad Targeting without Deep Packet Inspection

Gabriel Maciá-Fernández
University of Granada
Granada, Spain
gmacia@ugr.es

Yong Wang
University of Electronic Science
and Technology of China, China
ywang@uestc.edu.cn

Rafael Rodríguez
University of Granada
Granada, Spain
rodgom@correo.ugr.es

Aleksandar Kuzmanovic
Northwestern University
Evanston, IL, USA
akuzma@cs.northwestern.edu

***Abstract*—Online advertising is a rapidly growing industry currently dominated by the search engine 'giant' `Google`. In an attempt to tap into this huge market, Internet Service Providers (ISPs) started deploying deep packet inspection techniques to track and collect user browsing behavior. However, such techniques violate wiretap laws that explicitly prevent intercepting the contents of communication without gaining consent from consumers. In this paper, we show that it is possible for ISPs to extract user browsing patterns *without* inspecting contents of communication.**

Our contributions are threefold. First, we develop a methodology and implement a system that is capable of extracting web browsing features from stored non-content based records of online communication, which could be legally shared. When such browsing features are correlated with information collected by independently crawling the Web, it becomes possible to recover the actual web pages accessed by clients. Second, we systematically evaluate our system on the Internet and demonstrate that it can successfully recover user browsing patterns with high accuracy. Finally, our findings call for a comprehensive legislative reform that would not only enable fair competition in the online advertising business, but more importantly, protect the consumer rights in a more effective way.

I. Introduction

Online advertising is a $20 billion industry that is growing rapidly [1]. Examples of online advertising include contextual ads on search engine results pages, banner ads, rich media ads, social network advertising, online classified advertising, advertising networks, and e-mail marketing. Google [2], who originally controlled 35% of the ad server market recently acquired DoubleClick [3], a 34% market share holder, giving the combined online ad firm more than 69% of the market [4].

Internet Service Providers (ISPs) have for years looked on jealousy as Google has grown rich on their subscribers' web browsing, while the ISPs have been reduced to “dumb pipes”, ferrying internet traffic for subscribers but unable to win their online spending [1]. In an attempt to reverse this trend, some ISPs started cooperating with companies such as Phorm [5], NebuAd [6], and FrontPorch [7]. These companies use deep packet inspection techniques, *i.e.*, inspect a packet payload, to intercept web page requests and responses generated by ISPs' subscribers as they roam the net, and then apply behavioral ad targeting [8].

A major problem in the above arrangement between ISPs and companies that deploy deep packet inspection based data collection systems is a *legal* one:[1] unlike Google (not a broadband provider), ISPs that provide broadband services are *not* exempt from the Federal Wiretap Act, originally enacted in 1968 to protect against phone wiretapping and amended in 1986 to cover computer network communications. It states a simple prohibition: *thou shalt not intercept the contents of communications (see 18 U.S.C §2411(1))* [11]. *Violations can result in civil and criminal penalties.* Indeed, this prohibition has clearly been violated by deep packet inspection techniques [12]. The law predicts several exceptions, *e.g.,* security reasons (see 18 U.S.C. §2511(2)(a)(i) [11]) or user consent (see 18 U.S.C. §2511(2)(d) [11]) but behavioral advertising is certainly nowhere on the exception list.

Pressed by the legal constraints on one side, and by huge market opportunities on the other, ISPs (*e.g.*, [13], [14]) started addressing the legal issue by altering their customer-service agreements to permit monitoring by describing it as “performance advertising services” [1]. Each company allows users to opt out of the ad targeting, though that permission is buried in customer service documents' footnotes. Still, there is a strong concern that these approaches opt the user out of targeted ads, but *not* the online data collection. Hence, there is a fear that ISP-enabled ad targeting with deep packet inspection techniques is highly vulnerable to lawsuits [15], which is why many ISPs are reluctant to deploy this technology.

The key finding of this paper is that it is possible to design a methodology and a system that would make ISP-based user tracking and behavioral advertising *legal*, even without user consent. Leaving for the moment many important implications of this finding (that we discuss at the end of this section), we argue that such a system could be a 'game changer' in the contention between established Web-based and emerging ISP-based behavioral advertisers: it can remove the legal concerns that currently fundamentally constrain ISP-based behavioral ad targeting.

The main idea lies in abandoning controversial deep packet inspection techniques and reverse engineering user browsing patterns using alternative methods. We refer to the Electronic Communications Privacy Act [16] which defines the sharing of particular types of stored records of online activities. It states that any provider can hand-over *non-content* records to anyone except the government (see 18. U.S.C. §2702(c)(6)

This work is supported by Spanish MEC project TEC2008-06663-C03-02 (70% FEDER funds), NSF CAREER Award no. 0746360, and China Scholarship Council.

[1] In this paper, we focus on the U.S. Federal Law. Still, many international laws are similar to the U.S. Federal law [9], since laws have been harmonized through treaty and convention [10].

[16]). Consequently, sharing non-content-based stored headers — such as TCP headers — with anyone except a government body is legal [9].

The key challenge and the main research question we attempt to address then become if it is possible, and how accurately and scalably, to recover user browsing access patterns based solely on fairly limited information provided in TCP headers? We demonstrate that web browsing patterns stay highly visible at the TCP layer, and we design a method to automatically extract such features. Next, we profile the websites by extracting relevant features such as object size, cacheability, location, link information, transfer modes *etc*. Finally, we design an algorithm that correlates the two sources of data to detect the pages accessed by clients.

We extensively evaluate our algorithm and show that it achieves high detection rates, *i.e.*, 86%, with false positive rates below 5%. The fundamental reason for such performance is its ability to extract and exploit significant statistical page diversity available at all sites we explored. Most importantly, we demonstrate that the algorithm is resilient to data staleness, *i.e.*, when either network traces or web profiles are outdated. While the page properties necessarily change over time, we show that a subset of unique properties remain, making the detection resilient with time.

We further show that the approach is resistant to different browsing scenarios including pipelining, caching, NAT-level flow multiplexing, and various browser versions. We also demonstrate that the algorithm scales to entire websites while preserving high detection performance. Finally, we evaluate our approach in the ’wild’ and successfully recover browsing patterns based on real traces collected from a group of 17 volunteers.

II. Recovering Web Browsing Patterns from Stored TCP Headers

In this section, we introduce a methodology for recovering web browsing patterns from the information available in TCP headers. First, we briefly present the necessary background on the topic. Next, we describe our approach and the corresponding algorithm. Finally, we discuss a method for handling several possible sources of error.

A. Background

A web page typically consists of a *root file* and corresponding object files. A root file is uniquely determined by a URI [17]. A root file *references* other object files, or *objects*, that a given web page consists of , *e.g.* images, scripts, etc. If an object resides at the same server (determined by a unique IP address in our approach) as the root file, we term the object as *internal*. Otherwise, it is *external*.

Caching is another important Web-related mechanism. It allows retrieving web page objects from intermediate repositories such as proxies, shared caches, or browsers. Browsers and servers have mechanisms to decide if a given object should be cached or not [18]. Hence, objects in a web page could be *cacheable* or *non-cacheable*.

In addition to referencing objects, a root file typically hosts *links* (pointers) to other web pages as well. They enable clients to access other web pages “by clicking” them at a given web page. In our approach, *a given URI corresponds to a root file if and only if there exists a link on the given web site pointing to the given URI.* Like objects, links could be external and internal depending on the location of the corresponding root file’s URIs.

B. Methodology

The problem we aim to solve is the following: “Given a packet-level network trace, recover the web pages visited by users *without* ’touching’ the packet payload.” The key idea is as follows: (i) profile Internet websites visited by users represented in the trace, *i.e.*, independently crawl the given websites and collect statistics about web pages, *e.g.*, object size, cacheability, locality, links among pages, *etc*.; (ii) extract the Web-level communication features from the network-level information available in TCP headers; and finally, (iii) correlate the information from the two sources to detect web pages actually accessed by clients.

1) Website Profiling: To accurately and comprehensively profile a website, we develop a web crawler, which extracts the following characteristics about the web pages at a website that will later be used to recover actual user access patterns at that site.

Size. A root file or an object corresponding to a given URI could be downloaded in either *plain* or *compressed* mode, depending on browsers and servers settings. Our goal is to obtain the corresponding file size (in bytes) in both modes.

Cacheability. The HTTP response header obtained from the server for a given URI allows the crawler to estimate if a root file or an object is cacheable or not.

Locality. The crawler records the location of each root file and its corresponding objects. The location could be internal, *i.e.*, the root file or the object is hosted at the same server (same IP address) as the website. Otherwise, the location is external.

References. A root file pointed to by a URI could contain references to other URIs, corresponding to objects. The crawler parses this root file and extracts a list containing these references.

Links. A root file pointed to by a URI could contain links to other URIs. The crawler parses this root file and extracts a list containing these links. It further crawls URIs corresponding to internal links, *i.e.*, root files.

2) Extracting Web Browsing Features from Network Traces: Here, our goal is to extract the Web-level browsing features from network traces; in particular, the number of web pages accessed by a client and the size and location of root files and objects corresponding to these web pages. When combined with the information obtained via web profiling (Section II-B1), these features will enable recovering user web browsing patterns (Section II-B3). We refrain from mining packets payloads to obtain URIs accessed by clients or content generated by servers since both approaches violate the Federal Wiretap Act [9]. Indeed, we constrain ourselves to recording and later inspecting TCP headers only.

One-way TCP header collection. Our approach is tailored towards access ISPs, and it requires an ISP to record TCP packet headers at a tapping point in the network. While it is generally possible to obtain data in both (client-server and server-client) directions in access networks, that is typically not the case in non-access networks due to path asymmetry [19]. Still, our approach is applicable even in such scenarios because it requires collecting TCP headers in a *single* direction only, *i.e.*, from clients to servers, as we explain in detail below. To extract HTTP level communication from the trace, we filter out traffic on port 80 and create per source IP subtraces.

Webpage-based trace slicing. Our next goal is to further separate each of the user subtraces into separate *slices*; each slice should ideally consist of packets that correspond to a single page accessed by a client. To achieve this goal, we exploit the well-known Web-user behavior. In particular, it has been shown experimentally that either a machine [20] or a Web user [21] requires at least one second to process and react to the display of a new page. (We experimentally verify this result ourselves in Section IV by evaluating a representative user browsing data set we collected.) As a result, each user 'click' at a link on a website is followed by a period of activity corresponding to a web page download, and a period of inactivity corresponding to the page processing. Hence, we use these moments of user inactivity to separate the user traces into slices. Even when this is not the case that more than one page can end up in a slice, our algorithm can handle this situation as well, as we demonstrate below.

Extracting web page features. Our next goal is to extract the size and the position (internal vs. external) of the root file and the objects corresponding to a given web page associated with a given slice. To recover these features, we inspect the TCP packet headers corresponding to the given slice.

Three issues are considered here. First, when accessing a web page, the corresponding root file is always requested and downloaded first. Second, each HTTP request for any of the web page objects is requested in a *separate* set of TCP packets except when pipelining is enabled (Section II-C2). Third, in the vast majority of scenarios, TCP packets carrying HTTP requests have the TCP PUSH flag set.[2]

To verify this behavior at a large scale, we analyze network traces that we obtain from two Tier-2 ISPs from two different parts of the world. In one of the traces, we collect 131,681 HTTP requests, and in the other one we collect 153,853 requests. In the first case, in more than 96% of scenarios TCP packets corresponding to HTTP requests have the TCP PUSH flags set. In the second case, more than 95% cases have the TCP PUSH flags set on.[3]

Next, to estimate the root file and objects sizes within a trace slice, we proceed as follows. We consider that the TCP packets corresponding to a root file or an object are those belonging to the same TCP connection and are delimited by two consecutive TCP PUSH enabled packets. Once the root file and different objects contained in every slice have been identified, we extract their sizes from the acknowledge numbers available in TCP ACK packets. Finally, we determine the object location, *i.e.*, internal vs. external, in a straightforward way by checking IP address of the corresponding server.

TABLE I
SUMMARY OF THE STEPS IN THE DETECTION ALGORITHM.

```
TAGGING PHASE
1) For all $E_k \in E$:
    a) For all $R_i \in R$, check if $S(E_k) = S_m(R_i)$ and $L(E_k) = L(R_i)$:
        i) If true→ $R_i$ = identified.
    b) If only one $R_i$ = identified→$R_i$ = unique.
    c) For all $O_{ij}$, check if $S(E_k) = S_m(O_{ij})$ and $L(E_k) = L(O_{ij})$:
        i) If true→ $O_{ij}$ = identified.
    d) If only one $O_{ij}$ = identified→ $O_{ij}$ = unique.
2) Build sets $P_R$ and $P_O$ with pages with identified root files / objects respectively.
SELECTION PHASE
1) Initial set of detected pages: $P_D = \emptyset$
2) If pages in $P_R \cup P_O$ contain any unique root files/objects, $P_D$ = pages in $P_R \cup P_O$ with any unique root files/objects. End of algorithm.
3) $P'_D = P_R \cap P_O$. If $P'_D = \emptyset \rightarrow P'_D = P_R \cup P_O$.
    a) If $S(P'_D) = 1$, $P_D = P'_D$. End of algorithm.
4) Obtain $P''_D$ selecting from $P'_D$ those pages with highest percentage of identified objects.
    a) If $S(P''_D) = 1$, $P_D = P''_D$. End of algorithm.
5) Obtain $P'''_D$ filtering $P''_D$ with link information.
    a) If $S(P'''_D) = 1$, $P_D = P'''_D$. End of algorithm.
```

P_O: Pages with identified objects, P_R: pages with identified root files, E: set of all elements in the trace, E_k: element k in trace, R: set of all root files in the website, R_i: root file i, O_{ij}: object j in page i, $S_m(X)$: size type m of root file/object X, $S(\cdot)$: size operator, $L(\cdot)$: location(internal/external) operator, $\emptyset$:empty set.

Several issues, including the ability to estimate object boundaries (*e.g.*, due to pipelining) and the file size estimation accuracy (*e.g.*, due to variable HTTP header size), exist. We analyze these issues in depth in later parts of the paper (Sections II-C and III-E).

3) Detection Algorithm: Here, we present an algorithm that correlates information obtained via website profiling (Section II-B1) and features independently extracted from TCP headers (Section II-B2) with the goal of detecting actual web pages accessed by clients in the trace. The algorithm is independently executed on each *slice* of the trace. To avoid confusion, we refer to root files and objects identified in the trace as *elements*.

Denote by $E = \{E_1, E_2, ..., E_l\}$ the set of l elements identified in a trace slice. Next, denote by $P = \{P_1, P_2, ..., P_n\}$ the set of n web pages identified at the given website in the web profiling phase. Further, denote by $R = \{R_1, R_2, ..., R_n\}$ the set of n root files associated with the identified web pages. Also, denote by $O_i = \{O_{i1}, O_{i2}, ..., O_{im}\}$ the set of m different objects contained in page P_i. As we explained above, each root file R_i or object O_{ij} can be downloaded in either plain or compressed mode. As a result, we have two possible values for the size of each of the root files or objects. In particular, for a given root file or object X, denote by $S_1(X)$ its size in the plain mode, and by $S_2(X)$ its size in the compressed mode. Finally, the goal of the algorithm is to determine a subset of detected pages P_D, $P_D \subset P$. The algorithm is executed in the following two phases, also summarized in Table I:

Tagging phase. During this phase, for each element E_K

[2] Even when TCP PUSH flag is not set in the TCP header, HTTP requests could be distinguished based on the TCP packet size, which is greater than the TCP ACK size.

[3] The root cause for not achieving 100% is the existence of POST REQUESTS which may have a long size and are split into several packets with PUSH flag activated.

from the set of identified elements E, we compare the size $S(E_k)$ and the location $L(E_k)$ (internal/external) of the elements in the trace slice separately with the size and location of all root files and then objects in the website profile. Note that considering the location of an element as internal/external instead of its server IP address permits the tagging of objects which are downloaded from CDN networks. This allows us to identify possible candidate web pages to be selected as downloaded. Each root file or object whose size corresponds to that of one of the elements is tagged as *identified* (Table I, tagging phase, steps 1(a) and 1(c)). Moreover, if E_k is identified with a single root file or a single object in their respective comparisons, it is also tagged as *unique* (Table I, tagging phase, steps 1(b) and 1(d)). Because a unique object/root file is present in only one page of the website, its identification makes this page a good candidate to have been downloaded. Finally, in this phase, all the pages with identified root files are compiled in a set P_R and those with identified objects in a set P_O.

Selection phase. The selection phase takes the set of pages P_R and P_O as input data and aims to decide which pages are downloaded in the trace, and hence should be included in P_D (initially empty, Table I, selection phase, step 1). We distinguish two different cases: (i) if unique root files and objects have been identified in the slice, all the pages that contain them are selected (Table I, selection phase, step 2). Indeed, because multiple web pages might be present in the slice (Section II-B2), selecting pages with unique characteristics leads to high detection rates in such scenarios, as we will demonstrate in Section III-E3 below. (ii) However, in case that no unique root files or objects are identified, we make a best effort to minimize false positives; hence, our goal is to identify a *single* page in the slice.

We apply the following strategy. First, we consider only those pages, if any, that are present in both P_R and P_O; that is, $P_R \cap P_O$. Indeed, if there is an overlap between the two sets, it is likely that a page from the overlap has been accessed. However, if there is no overlap, we are unable to reduce the set, and hence we consider all the pages in both sets $P_R \cup P_O$. The resulting set is P'_D (Table I, selection phase, step 3). If more than a single page still remain, we filter P'_D and extract only page(s) with the highest percentage of identified objects, *i.e.*, set P''_D (Table I, selection phase, step 4).

If several candidates still remain, we consider the user navigation pattern. In particular, we use the simple heuristic that if a user accesses more than a single page at a website, it is likely that there exist links from one page (hence one slice) to the next page (hence next slice) accessed by the client. Thus, among the remaining candidates in ith slice, we only choose those that are linked from the pages in the P_D set obtained for the previous $(i-1)$th slice. The resulting set is P'''_D, (Table I, selection phase, step 5). During steps 3–5, if any set P'_D, P''_D or P'''_D contains only one page, it is selected as the final decision P_D (Table I, selection phase, steps 3(a), 4(a), and 5(a)). Otherwise, if several candidates still remain, they are all discarded in order to minimize the false positives.

C. Dealing with Sources of Errors

Here, we emphasize the key factors responsible for false detection. First, we summarize the key elements that lead to inaccuracies in a web object size estimation. Then, we outline other factors that can impact detection accuracy. We evaluate all these factors and their impact on the detection accuracy in Section III.

1) Object Size Estimation: The estimate of an object (or a root file) size obtained (i) via website profiling (Section II-B1) and (ii) via TCP-level headers (Section II-B2) can be different. Whenever such a difference occurs, the probability that the algorithm will make a false decision increases. The key factor contributing to the difference in the estimated object size is the potential variability in the HTTP header size. We provide several examples below.

First, an HTTP request may include a cookie. Although the size of a cookie is usually constant, in some cases its length depends on a seed used for its generation, which might involve parameters such as nonces, timestamps, or source IPs. In order to reduce the amount of false positives due to cookies, our crawler considers two different sizes for those pages that return a cookie: one taking into account the cookie size and another without it.

Second, an object might be downloaded using the chunking transfer mode [18]. Indeed, when a server does not know in advance the total size of the content that it is sending, the sender breaks the message body into chunks of arbitrary length, and each chunk is sent with its length prepended [18]. Hence, the complete size of the object depends on the number of chunks used and their own size. As a result, it can happen that subsequent requests to the same non-cacheable objects on the same site can generate different HTTP header sizes.

2) Other Sources of Error: Here, we outline other factors that can lead to detection inaccuracies.

Dynamic website behavior. Websites can change over time. For example, a site administrator can modify the content of a given page. The relevant question thus becomes: How frequently do root files or objects at a website change, and how does that affect the ability of the algorithm to detect such pages? We explore this issue in depth in Section III-D below.

Pipelining. HTTP1.1 proposes pipelining, *i.e.*, send subsequent HTTP requests within a single TCP connection without waiting for the corresponding HTTP responses. This approach blurs the visibility of object boundaries at the TCP level and complicates the corresponding web object size estimation. While pipelining is not widely spread in the Internet, as we demonstrate later in the paper, a relevant question is how our algorithm performs when pipelining is enabled. We explore this issue in Section III-E1.

Caching. All objects belonging to a page are not always downloaded from the server. While we explicitly address this issue in the algorithm, the question is how does this mechanism affect its accuracy We explore this issue in Section III-E2.

Overlapping page downloads. Several factors can generate so-called overlapping page downloads to appear in a single trace slice. First, inter-click estimation might not always be fully accurate. Hence, it can happen that two or more web page downloads from the same website can end up in the same trace slice. Second, when Network Address Translation (NAT) boxes are used, a number of clients behind the NAT will have the same source IP address visible at the tapping point. While accurate per-client trace slicing is still feasible using destination (server) IP addresses, it is possible that at given time intervals, one or more clients behind the NAT concurrently access the same website. Third, a single user can (nearly) concurrently access several pages at a single website. All these issues lead to the overlapping page downloads effect. We explore our algorithm's performance in such scenarios in Section III-E3.

Spurious requests. During the navigation process, certain spurious HTTP requests that do not correspond with a page download can be generated. These are mainly caused by client web-based applications, *e.g.*, Google toolbar, live search toolbar, or by AJAX scripts embedded in web pages. In some cases, these requests can be filtered by considering the usual destination IP addresses, *e.g.*, Google server. AJAX scripts, on the other hand, are a well-known challenge even for the latest commercial crawlers. Hence, these requests will interfere with the detection process generating false positives.

III. Evaluation

Here, we evaluate our approach in a number of challenging, yet realistic scenarios. In particular, we explore the resilience of our algorithm when either a web profile or a network trace is outdated. Then, we explore the issues of pipelining, caching, overlapping page downloads, and the browser diversity.

A. Experimental Setup

Before presenting the performance evaluation, we first explain how we obtained two necessary datasets – crawled website logs and TCP-level traces. To emulate a realistic setup, in which the two datasets are typically obtained from two different points in the network, (*i.e.*, TCP-level traces collected from an ISP network, and crawled logs by a different set of machines). In all scenarios, we use a crawling spider we designed to profile the websites; we generate network traces using the `Firefox 3.0.5` browser, with default parameters, *i.e.*, caching enabled and pipelining disabled. We explore other browsers and parameter settings in Section III-E4 below.

Website profiling. We select six representative websites, which are The New York Times (www.nytimes.com), FC Barcelona (www.fcbarcelona.com), IKEA (www.ikea.com), Toyota (www.toyota.com), Northwestern University (www.northwestern.edu, Univ1), and University of Granada (ceres.ugr.es, Univ2). Some of them adopt CDN techniques, e.g., Nytimes, IKEA, and Toyota; while others host their content by themselves. While this is certainly a small fraction of the Web, our key goal is to understand in-depth performance of our algorithm in diverse scenarios in which either CDN is involved, or web profiles are outdated, or TCP traces are stale (See details in Section III-D). In the next section, we perform experiments in the 'wild' and evaluate our algorithm by crawling a larger number of websites.

In each of the sites, we crawl a subset of pages, *i.e.*, 2,000 web pages (except for Univ2 which has less than 2,000 pages). We select this threshold because it enables us to crawl all six websites within a 24 hours interval. This helps us to understand how our algorithm performs when either a web profile or a network trace is outdated, an issue that we explore in depth in Section III-D below. Finally, in order to understand how the size of a website impacts the results, we crawl one of the websites in full in Section III-F.

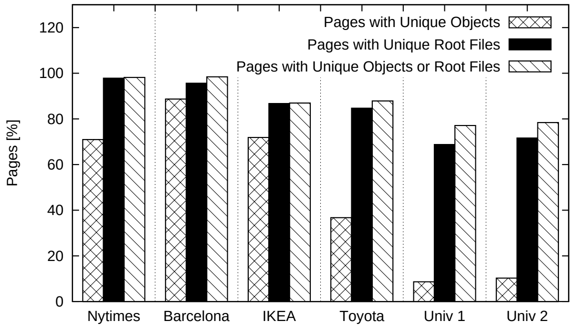

Fig. 1. Sites statistics

TCP-level traces. To obtain TCP-level traces (using the Wireshark tool [22]), we emulate user behavior by creating quasi-random walks over 100 out of the 2,000 pages at a website; we call these 100 pages *test pages*. In particular, we at random select a page out of the 2,000 pages; then, we randomly select the next page from the set of pages that the given page links to. When no links exist from a given page, we randomly select another page from the set, and continue the quasi-random walk until we collect 100 pages. Finally, we compute detection statistics as we explain in Section III-C below. For all experiments, we collect ten independent test sets, and show averages. We move beyond emulation in Section IV and deal with real user browsing traces.

B. Site Uniqueness

Here, we show the statistics for unique-size root file and object in the six websites. Such files are invaluable in the detection process since their presence in a trace uniquely identify a web page.

Figure 1 shows the percentage of pages with unique objects, unique root files, and with either unique objects or unique root files. The figure shows that the percentage of pages with unique objects is high, except for the two universities. This is because commercial or news websites are usually rich with pictures and other objects, which dramatically increase the page diversity. For example, in the IKEA website, many pages have a unique picture showing different products.

The figure shows that the percentage of pages with unique root files is enormous in all web sites. Indeed, even when the web pages share the same template, they still have different text resulting in different root file sizes. Moreover, the percentage of pages that either have unique size objects or unique root files is necessarily even higher. These high percentages indicate that the use of unique-size objects or root files is a powerful feature.

We use these statistics to explain the basic performance of our algorithm. In particular, from the statistical point of view, the percent of pages with unique objects could be considered as the (loose) lower bound of the expected success rate, and

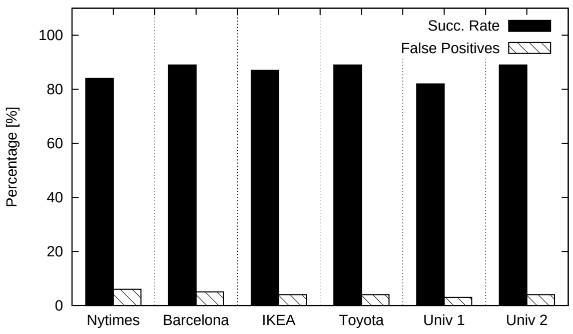

Fig. 2. Basic performance of the detection algorithm

the percentage of pages with unique size objects or unique root files is considered as the (loose) upper bound.

C. Basic Performance

Here, we explore the performance of our algorithm for the six websites. We apply the methodology explained in Section III-A above, *i.e.*, use 100-page long test sets to compute the success rate, false positives and negatives. In the figures here and in the rest of the paper we show the success rate and false positives. (False negatives could be computed as 100% - success rate (%)).

Figure 2 shows the results. We make several observations. First, the success rate is around 86% on average over the websites, and false positives are below 5%. In all cases, the success rate is above the lower expected bound, as we predicted above. Moreover, in certain scenarios (IKEA, Toyota, Univ1, and Univ2), the performance is even above the upper expected bound. This is because we made expectations only based on the site uniqueness. Still, other issues, such as the use of link information, can further improve the results even in scenarios when no unique items are detected at a website.

The performance for Barcelona and Nytimes is approximately between upper and lower bounds. In both cases the reason for not reaching the upper-bound performance is due to effects explained in Section II-C1. In particular, we experienced increased chunking-mode transfers in the Nytimes case for root files. Nevertheless, other factors, such as unique objects, the percent of identified objects and link relationships, keep the performance above the lower expected bound. As an example, in the Nytimes case, success rate of 84% (Figure 2) surpasses the lower expected bound of 71% (Figure 1).

D. The Role of Time Scales

Both network traces and web profiles could be outdated for a number of reasons. For example, several days might pass until an ISP ships its traces to an advertising company. Likewise, crawling the Web is an exhaustive process. Hence, several days or more can pass until a crawler revisits a site and updates its profile. Here, we evaluate how these issues impact the accuracy of our algorithm.

Methodology. We select 100 pages as the *preliminary test set* for each website in the first day of the experiment. Then, we crawl the given sites once a day for one week, and collect a new 2,000 pages profile each day for each of the sites. Because some of the websites change over time, the 2,000 pages that we obtain are not always the same. As a result, the initial test set also reduces in some cases. Although the pages crawled on the first day typically still exist on the website, our limited crawling process does not manage to download these pages. Hence, we proceed in two steps. First, we determine the pages that exist during the entire period and consider them as a *final test set*. Second, we explore how the pages in the final test set change over time.

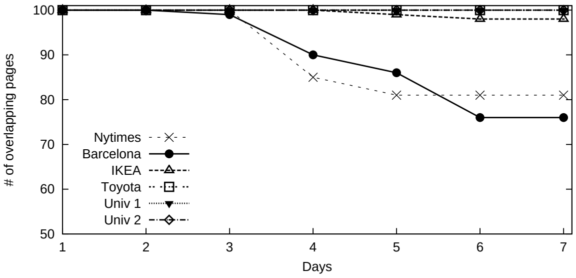

Fig. 3. Evolution of the number of overlapping pages during a week.

Figure 3 illustrates that, during the seven day period, the number of overlapping pages stays the same in Toyota and universities, suddenly dives a bit in Nytimes and Barcelona, and gradually decreases in IKEA. Again, all the web pages from the first day are typically available on the web site, the overlap decrease is due to the limited number of crawled pages and the addition of new pages. More specifically, Toyota's updates are relatively the slowest as its products are usually coming out over longer time scales. On the other side, Nytimes may add many pages in its website in one day, which leads to a huge shrink in the overlapping size. IKEA, as an in-between case, slowly updates its website and hence the number of overlapping pages decreases at the same pace. As a result, for Nytimes, Barcelona, and IKEA cases, the size of the final test set is 81, 76, and 98 pages respectively, while for Toyota and universities cases, the size is 100 pages.

Finally, we divide the six websites in two categories. The first one includes sites that have the final test set less than 100 pages (Nytimes, Barcelona, and IKEA). For this set, we capture the TCP-level trace at the last day of the experiment (day 7 in Figure 3). The second set includes sites that have the final test set equal to 100 pages (Toyota and universities). For this set, we capture the TCP-level trace at the first day of the experiment (day 1 in Figure 3). In the former scenarios, the website profiles are out of date. In the latter scenarios, the TCP-level traces are stale. In the experiment, we compare the TCP-level traces with web profiles taken during the seven day period.

Performance. Figures 4(a) and 4(b) show the success and the false positive rates (computed over the final test) as a function of time. The reference point in each figure (day 0) corresponds to the time when TCP-level traces are obtained. As a result, day 0 in Figure 4(a) corresponds to day 1 in Figure 3. Likewise, day 0 in Figure 4(b) corresponds to day 7 in Figure 3.

Figure 4 provides three insights. First, in Toyota and the universities (Figure 4(a)), the success rate stays almost constant; in other scenarios (Figure 4(b)), the success rate changes marginally. For example, the success rate of Barcelona drops

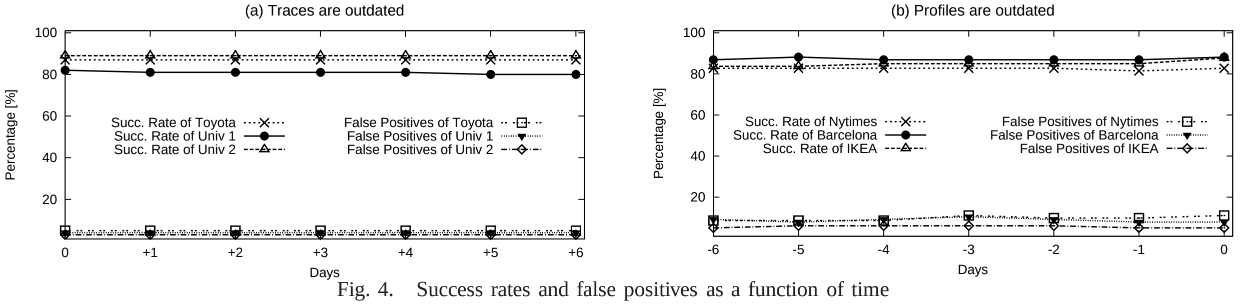

Fig. 4. Success rates and false positives as a function of time

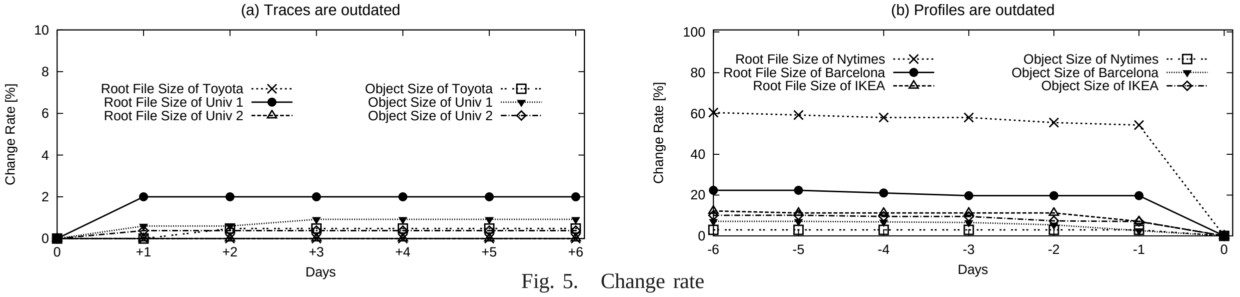

Fig. 5. Change rate

from 88% to 87%. Second, in all cases, the success rates reach the peak on the day when the test TCP-level trace is collected because the properties of root files and objects in the crawled profiles are more likely to be the same as those in the TCP-level trace on the same day. Third, besides the success rates, false positives are also resilient with time. In Figure 4(a) the change rate of false positives remains same; In Figure 4(b) the false positives change smoothly; for example, in the IKEA case, the minimum is 5% and the maximum is 6%.

Change rates. To understand the causes of the above observations, we explore the change rates of root files and objects size. In the experiment, we compare the root files and objects size of pages in the final test set with web profiles taken during the seven day period. We define the change rates as the percentage of inconsistency of root files or objects size between the final test set and web profiles. We also consider the objects that are permanently removed from the given pages as changed, *i.e.*, their size becomes zero.

Figure 5 shows the root file and object change rate as a function of time. The first finding is that the change rate of both root files and objects is much smaller in Figure 5(a) than in Figure 5(b). This is caused by the same reasons discussed with respect to Figure 3 above. For example, in the Toyota case, the web administrators update their web news if some new products are available in the market, which typically happens over longer time scales. On the contrary, the websites are updated much more frequently for news and other commercial websites such as Nytimes, Barcelona, and IKEA.

Second, the change rate increment is the largest within one day from when the traces are taken, *i.e.*, day +1 in Figure 5(a) and day -1 in Figure 5(b). After that, there is almost no change, *i.e.*, for days 2 – 6 in Figure 5(a) and days -2 – -6 in Figure 5(b). This is because a part of pages, like main pages, updating the top-line news or the latest product promotions at commercial websites is typically updated frequently, not all the pages.

Finally, in all cases, the change rate of root files is higher than the objects size change rate. For example, In Nytimes case, the root files change rate of 60% highlights the above fact that news web pages, particularly the text part, are frequently updated. At the same time, the change rate for objects is less than 3% at day -6. Thus, despite a highly dynamic site behavior, our algorithm is capable of accurately detecting the given web pages with high accuracy, as we demonstrated in Figure 4. This is because a subset of web pages' unique properties remain consistent over time.

E. Different Browsing Scenarios

Here, we explore different browsing scenarios. Thus, we evaluate how (i) pipelining, (ii) caching, (iii) overlapping page downloads, and (iv) different browsers affect the performance. The first three experiments thus far are conducted using the `Firefox 3.0.5` browser with its default settings, *i.e.*, caching enabled and pipeli-ning disabled by default. We conduct all experiments on the Toyota server, using the above methodology. To avoid dynamic effects explored above, we collect all traces on the same day.

1) Pipelining: We first explore how widely pipelining is spread in the Internet by analyzing a Tier-2 network trace with 153,583 HTTP requests. We identify the existence of several HTTP requests in a same TCP segment as a pipelining signature. Our results show that the percentage of pipelined segments is smaller than 1%, while the percent of users that use browsers with pipelining enabled is around 2% of the total number of users (in terms of source IP addresses).

Despite low usage of pipelining, clients might be tempted to enable this feature in order to prevent ISP-based ad targeting. We explore whether such an attempt would be successful.

Table II shows the results. We can see that there is only a slight difference in the results, as the success rate degrades by 1% only. The reasons are the following. First, the fact that a browser enables pipelining does not imply that all HTTP

TABLE II
PERFORMANCE EVALUATION FOR DIFFERENT BROWSING SCENARIOS

Scenario	Success rates	False positives
Pipelining disabled	89%	4%
Pipelining enabled	88%	4%
Cache disabled	90%	4%
Cache enabled	89%	4%
Sequential	89%	4%
Parallel-two	74%	7%
Parallel-four	63%	8%

requests will be pipelined (for performance reasons), but only a subset of them. Indeed, only 12% of the TCP segments containing HTTP requests are really pipelined by the browser. As a result, the bulk of the objects sizes are correctly identified. Second, even if larger percents of objects would be pipelined, the requests for root files cannot be pipelined. This is because a browser does not know in advance which objects to fetch before it downloads the root file. Hence, high detection rates are still feasible.

2) Caching: In this experiment we evaluate the effects of browser caching mechanisms. We consider two scenarios: (i) navigation without caching, in which we disable the cache in the browser and (ii) navigation with caching, in which we enable the cache in the browser.

Table II shows the results for the two scenarios. As expected, we can see that results when cache is disabled are better, *i.e.*, the success rate increases to 90%, while false positives remain unchanged relative to the caching scenario. The slight improvement in the performance is due to the fact that the existing non-cacheable elements (typically all root files and a subset of objects) already create a strong inter-page diversity. Nevertheless, more information in the non-caching scenario produces a better result.

3) Overlapping Page Downloads: Overlapping page downloads means that more than one web page might end up in a single trace slice. For example, this can happen either due to NAT-induced effects or inaccurate inter-click time estimation. While we show in the next section that none of the two effects are likely to happen, we nevertheless explore our algorithm's performance in this case. For this, we have emulated the download of a test set of pages with three different navigation patterns: (i) pages have been downloaded without overlapping (*sequential browsing*), (ii) two different pages are downloaded simultaneously (*parallel-two browsing*), and (iii) four different pages are downloaded simultaneously (*parallel-four browsing*).

Table II shows the results. As expected, the performance is the best in the sequential case, when there is only a single page in a slice. While the success rate necessarily degrades when the number of pages increases per slice, it is still quite reasonable (74% in parallel-two and 63% in parallel-four). These results are mainly due to the step 2 of the selection phase (Table I), which takes advantage of unique root files and objects from multiple pages.

4) Different Browsers: We experiment with different browsers. In particular, we obtain different traces using Firefox 3.0.5, Internet Explorer 7.0, and Google Chrome. All the browsers disable pipelining and enable caching. We have not found any differences in the performance of the algorithm when using the three traces obtained. This implies that our approach is independent from different browser types.

F. Scaling the Website Profile

To evaluate how our approach behaves with increased website profile, we crawl the entire Toyota site and download 9,211 pages. Then, we repeat the experiment by repeating the procedure explained above.

Our results show that the success rate is resilient with the increase of the website profile. More specifically, the success rate of Toyota reduces from 89% to 81%. At the same time, the false positives increase from 4% to 7%. We investigate this result in more depth, and find that 78% of pages have either unique size objects or unique root files, while this percentage was about 88% when the website profile was 2,000 pages long (Figure 1). Additionally, each page in Toyota site has 97.3 links on average which reduces the ability of our algorithm to sweep out many incorrect results.

IV. PERFORMANCE IN THE WILD

Here, we move away from the controlled environment in which we generate the test pages using quasi-random walks over the six websites. We evaluate our approach by using real user browsing patterns at websites of their own choice.

We collect URI-level traces from 17 volunteers (with their consent) from USA, Europe and Asia during 1 month. These traces contain users' anonymized identifications, as well as the visited URIs and their corresponding timestamps. From this information, we select 41 different websites with the highest number of requests. For each website we build its profile by crawling up to 2,000 pages. Then, we choose the list of URIs as the test set in our experiment. A TCP level trace is obtained by replaying the user navigation patterns (visited URIs and timestamps) within these sites.[4]

In this experiment we obtain a success rate of 85%. This result demonstrates that our approach indeed works well in the wild. Besides, the false positive ratio is 9%, slightly more than the result obtained in the controlled environment.

NAT behavior. To understand NAT-like behavior, we further explore URI-level traces from ten people from the same local network. We study the requests sent to the most popular website (a total of 16,756 requests) to discover the number of simultaneous accesses based on their timestamp. These simultaneous petitions generate what we have called overlapping pages downloads (Section III-E3). Considering that this happens when more than one user accesses the most popular site within the same second, we find that only 0.44% of the accesses are simultaneous. In summary, the presence of NAT boxes will not degrade the performance of our detection method since its impact is small.

Inter-click time. Finally, we verify the inter-click time statistics in order to validate our choice of 1 second for slicing the trace (Section II-B2). We process 297,885 timestamps in total and 94.53% of them have the inter-click time larger than 1 second.

[4] We compress all inter-access times longer than one minute to one minute.

V. Related Work

Our work relates to the security research efforts aimed towards analyzing and inferring encrypted web browsing traffic [23], [24], [25], [26]. The authors of these papers have demonstrated that it is feasible to reveal the sources of encrypted web traffic despite encryption. In light of this finding, they further analyze additional mechanisms that can help secure such communication. The key differences between our work and this thread of papers are three-fold. (i) We have shown that there are strong incentives to reveal user browsing patterns even when they are not encrypted. As a result, the scope of the problem changes from the one covering a small fraction of encrypted web pages on the Web to the *entire* Web 'landscape'. This dramatic change of scope in turn fundamentally impacts both (ii) our methodology and (iii) the range of potential counter mechanisms, as we elaborate below.

Regarding methodology, our approach differs from the security-oriented related work in three aspects. First, because we operate in the 'wild', unlike previous work, we consider multiple web features characteristic for 'open' web communication. This includes object location, uniqueness, cacheability, link information, different transfer modes, distinction between root files and objects, *etc.*, to characterize web pages. Second, we add mechanisms that consider possible sources of error that are inevitably created by the state-of-the art web practices (see Section II-C). Finally, contrary to previous work (*e.g.*, [26]) that has severe scalability issues, our approach can effectively scale. Indeed, we have demonstrated that it is feasible for an ISP to successfully collect TCP headers (and recover user browsing behavior) anywhere in the network, even behind a proxy or a NAT (Section IV). In addition, because the destination IP address is known in the advertising case, we effectively reduce the scalability problem from the entire web space to a *single web server*. Moreover, because we are capable of statistically characterizing a website in a more comprehensive way, we effectively scale the detection process.

VI. Conclusions

In this paper, we showed that it is possible to recover user web browsing patterns *without* inspecting the packet payload. By extracting HTTP-level 'reflections' available at the transport layer, and by profiling web sites in a comprehensive way using root files, objects, different transfer modes, linking information, cacheability, and locality, we designed an algorithm capable of effectively merging the two data sources and discovering web pages accessed by clients. We extensively evaluated our methodology on the Internet using both emulation and real user browsing patterns.

Our key insights are the following: (i) The development of the Web in recent years, *e.g.*, rich image mixtures and the use of CDNs, has created a significant statistical diversity among web pages at a website, making them highly identifiable. (ii) The page identifiability remains high even when a trace from an ISP is outdated, or when the web profile is not fresh due to crawling limitations. Even though the page features can dramatically change over time, we showed that a sufficient subset of identifiable features does stay available. (iii) The detection process is resilient to a number of challenges, including pipelining, caching, NAT-level multiplexing, different browser types, and it effectively scales. (iv) Endpoint-based countermeasures are highly limited; not only because it is hard to comprehensively cover rich inter-page diversity, but because websites have no incentives to apply such countermeasures since they are one of the primary beneficiaries of the advertising business.

In the broader context, we hope that this paper will open discussion in at least some of the topics below: (i) Internet advertising business needs more fairness, and our work here is a step in that direction, *i.e.*, it enables fair competition among different providers, independently from the type of service they are providing or their location on an end-to-end path. (ii) Consumer rights must be protected, and we argue that this can only be done via a modern legislative reform. (iii) The networking research community should not become a 'collateral damage' of such a reform; on the contrary, this could be an opportunity to specify data sharing practices for academic research in a more liberal way.

References

[1] "Washingtonpost.com: Every click you make," http://www.washingtonpost.com/wp-dyn/content/article/2008/04/03/AR2008040304052.html.

[2] "Google," http://www.google.com/.

[3] "Double click," http://www.doubleclick.com/.

[4] "Internet marketing news: Doubleclick deal means Google controls 69% of the online ad market," http://www.browsermedia.co.uk/2008/04/01/doubleclick-deal-means-google-controls-69/-of-the-online-ad-market/.

[5] "Phorm," http://www.phorm.com/.

[6] "NebuAd," http://www.nebuad.com/.

[7] "Frontporch," http://www.frontporch.com/.

[8] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen, "How much can behavioral targeting help online advertising?" in *WWW '09*.

[9] P. Ohm, D. Sicker, and D. Grunwald, "Legal issues surrounding monitoring during network research (invited paper)," in *ACM IMC '07*.

[10] "Convention on cybercrime, Budapest, 23.XI.2001," http://conventions.coe.int/Treaty/EN/Treaties/HTML/185.htm.

[11] "18 united states code 2511," http://www4.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002511----000-.html.

[12] "ISP behavioral targeting v. you," http://www.seoserpent.com/2008-09/isp-behavioral-targeting/.

[13] "Embarq," http://www.embarq.com/.

[14] "Wide open west," http://www1.wowway.com/.

[15] "Behavioral advertising could be illegal: NebuAd leaves ISPs vulnerable to wiretap, privacy laws," http://www.dslreports.com/shownews/94578.

[16] "18 united states code 2702," http://www.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002702----000-.html.

[17] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic syntax," Jan. 2005, Internet RFC 3986.

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

[19] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services," in *OSDI '06*.

[20] M. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, Dec. 1997.

[21] B. A. Mah, "An empirical model of HTTP network traffic," in *INFOCOM '97*.

[22] "Wireshark," http://www.wireshark.org/.

[23] H. Cheng and R. Avnur, "Traffic analysis of SSL encrypted web browsing," 1998.

[24] A. Hintz, "Fingerprinting websites using traffic analysis," in *Workshop on Privacy Enhancing Technologies '02*.

[25] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *ACM CCS '06*.

[26] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *IEEE Computer Society SP '02*.